UNIVERSIDADE DE SÃO PAULO

Escola de Artes, Ciências e Humanidades

Rosiane Correia Santos

LearnInPlanner: uma abordagem de aprendizado supervisionado com redes neurais para solução de problemas de planejamento clássico

LearnInPlanner: a supervised learning approach with neural networks to solve problems of classical planning

São Paulo

Novembro 2013

Rosiane Correia Santos

LearnInPlanner: uma abordagem de aprendizado supervisionado com redes neurais para solução de problemas de planejamento clássico

LearnInPlanner: a supervised learning approach with neural networks to solve problems of classical planning

Dissertação apresentada à Escola de Artes, Ciências e Humanidades da Universidade de São Paulo para obtenção do tıitulo de Mestre em Ciências. Programa: Sistemas de Informação.

Versão corrigida contendo as alterações solicitadas pela comissão julgadora. A versão original encontra-se disponível na Escola de Artes, Ciêencias e Humanidades.

Orientador(a): Profa. Dra. Patrícia Rufino Oliveira

São Paulo
Novembro 2013

Autorizo a reprodução e divulgação total ou parcial deste trabalho, por qualquer meio convencional ou eletrônico, para fins de estudo e pesquisa, desde que citada a fonte.

CATALOGAÇÃO-NA-PUBLICAÇÃO
Biblioteca
Escola de Artes, Ciências e Humanidades da Universidade de São Paulo

Santos, Rosiane Correia
LearnInPlanner : uma abordagem de aprendizado supervisionado com redes neurais para solução de problemas de planejamento clássico / Rosiane Correia Santos ; orientadora, Patricia Rufino Oliveira. – São Paulo, 2013.
75 f. : il.

Dissertação (Mestrado em Ciências) - Programa de Pós-Graduação em Sistema de Informação, Escola de Artes, Ciências e Humanidades, Universidade de São Paulo.
Versão corrigida.

1. Aprendizado computacional. 2. Inteligência artificial. 3. Redes neurais. 4. Algoritmos genéticos. 5. Solução de problemas. 6. Planejamento. I. Oliveira, Patricia Rufino, orient. II. Título.

CDD 22.ed. – 006.31

Dissertação de mestrado sob o título *"LearnInPlanner: uma abordagem de aprendizado supervisionado com redes neurais para solução de problemas de planejamento clássico"*, defendida por Rosiane Correia Santos e aprovada em 19 de Novembro de 2013, em São Paulo, Estado de São Paulo, pela banca examinadora constituída pelos doutores:

Profa. Dra. Patrícia Rufino Oliveira
Orientador

Profa. Dra. Leliane Nunes de Barros
Universidade de São Paulo

Profa. Dra. Karina Valdivia Delgado
Universidade de São Paulo

Dedico esse trabalho à memória de minha querida mãe, Ronilce Correia Santos, falecida em 20/04/1987, um dia após o meu nascimento. Uma pessoa a qual o destino tirou a oportunidade de realizar seus sonhos e pela qual sou motivada a nunca desistir dos meus.

Agradeço primeiramente à minha grande amiga Daiana, para os íntimos Doriana, por não me deixar perder a prova para ingresso no programa de mestrado. E também pelo seu incentivo, ombro amigo e apoio nos momentos de dificuldade.

À minha família por entender os momentos em que estive ausente para conseguir concluir este trabalho. E, em especial, à minha avó pelo exemplo de força de vontade e determinação que me ajudou a chegar onde estou.

À minha orientadora Patrícia Rufino Oliveira por todas as contribuições dadas ao longo do mestrado, sem as quais, não seria possível a conclusão deste trabalho. À professora Karina Valdivia Delgado pela ajuda fundamental no desenvolvimento deste trabalho.

Aos meus amigos da Ordem da Nutella (Roberto, Éric, Jadson, Leila, Renata, Fernando, Katatau) e amigos do mestrado, em especial Higor e Cláudio, por toda ajuda, sugestões, discussões, contribuições e momentos de descontração ao longo do mestrado.

Aos meus amigos do Colônia de Enzimas pelo apoio e por entenderem a minha ausência em muitas ocasiões devido ao mestrado. Em especial, deixo meu agradecimento à minha grande amiga Débora por sua amizade, compreensão, paciência em me ouvir, incentivo e por me ensinar a nunca deixar a peteca cair.

A todos os meus amigos do LARC e da TVoD, em especial ao Reinaldo, Fernando, Klava, Hélcio, Samuel, Lucas e Cindy, pelo incentivo em fazer o mestrado, pelos conselhos dados e por possibilitarem muitas vezes me ausentar para a realização deste trabalho.

A todos os amigos da academia de Kung Fu TSKF, em especial aos instrutores Thaís, Caio e Thiago, pelo apoio, ajuda em momentos que precisei e pelos ensinamentos aplicáveis não ao Kung Fu mas à vida.

Aos amigos que fiz no pouco tempo que trabalhei na empresa Insite (Camilla, Conde, Puga, Eduardo, Fábio e Andreia) pelo apoio dado durante a realização deste trabalho.

Aos amigos do LApIS pela oportunidade de trabalhar com vocês e pelo apoio dado para o meu mestrado.

Às demais pessoas, que porventura eu esqueci de citar aqui, e que contribuíram direta ou indiretamente com a realização deste trabalho.

Uma mente precisa de livros como uma espada precisa de uma pedra de amolar. Tyrion Lannister - *Game of Thrones*

Resumo

SANTOS, Rosiane Correia. ***LearnInPlanner*: uma abordagem de aprendizado supervisionado com redes neurais para solução de problemas de planejamento clássico**. 2013. 75 f. Dissertação (Mestrado em Ciências) – Escola de Artes, Ciências e Humanidades, Universidade de São Paulo, São Paulo, 2013

A busca progressiva no espaço de estados é uma das abordagens mais populares de Planejamento Automatizado. O desempenho dos algoritmos de busca progressiva é influenciado pela heurística independente de domínio utilizada para guiá-lo. Nesse contexto, o foco do presente trabalho consiste em investigar técnicas de aprendizado de máquina supervisionadas que possibilitaram agregar à heurística do plano relaxado, comumente utilizada em abordagens atuais de planejamento, informações sobre o domínio em questão que viessem a ser úteis ao algoritmo de busca. Essas informações foram representadas por meio de um espaço de características do problema de planejamento e uma rede neural MLP foi aplicada para estimar uma nova função heurística para guiar a busca por meio de um processo de regressão não linear.

Uma vez que o conjunto de características disponíveis para a construção da nova função heurística é grande, foi necessário a definição de um processo de seleção de características capaz de determinar qual conjunto de características de entrada da rede resultaria em melhor desempenho para o modelo de regressão. Portanto, para a seleção de características, aplicou-se uma abordagem de algoritmos genéticos.

Como principal resultado, tem-se uma análise comparativa do desempenho entre a utilização da heurística proposta neste trabalho e a utilização da heurística do plano relaxado para guiar o algoritmo de busca na tarefa de planejamento. Para a análise empírica foram utilizados domínios de diferentes complexidades disponibilizados pela Competições Internacionais de Planejamento. Além dos resultados empíricos e análises comparativas, as contribuições deste trabalho envolvem o desenvolvimento de um novo planejador independente de domínio, denominado *LearnInPlanner*. Esse planejador utiliza a nova função heurística estimada por meio do processo de aprendizado e o algoritmo de Busca Gulosa para solucionar os problemas de planejamento.

Palavras-chave: Planejamento Clássico, Aprendizado Supervisionado, Redes Neurais Artificiais, Rede Perceptron Multicamadas, Seleção de Características, Algoritmos Genéticos.

Abstract

SANTOS, Rosiane Correia. ***LearnInPlanner*: a supervised learning approach with neural networks to solve problems of classical planning**. 2013. 75 p. Dissertation (Master of Science) – School of Arts, Sciences and Humanities, University of São Paulo,São Paulo, 2013.

The forward state-space search is one of the most popular Automated Planning approaches. The performance of forward search algorithms is affected by the domain-independent heuristic being used. In this context, the focus of this work consisted on investigating techniques of supervised machine learning that make possible to agregate to the relaxed plan heuristic, commonly used in current planning approaches, information about the domain which could be useful to the search algorithm. This information has been represented through a feature space of planning problem and a MLP neural network has been applied to estimate a new heuristic function for guiding the search through a non-linear regression process.

Once the set of features available for the construction of the new heuristic function is large, it was necessary to define a feature selection process capable of determining which set of neural network input features would result in the best performance for the regression model. Therefore, for selecting features, an approach of genetic algorithms has been applied.

As the main result, one has obtained a comparative performance analysis between the use of heuristic proposed in this work and the use of the relaxed plan heuristic to guide the search algorithm in the planning task. For the empirical analysis were used domains with different complexities provided by the International Planning Competitions. In addition to the empirical results and comparative analysis, the contributions of this work involves the development of a new domain-independent planner, named *LearnInPlanner*. This planner uses the new heuristic function estimated by the learning process and the Greedy Best-First search algorithm to solve planning problems.

Keywords: Classical Planning, Supervised Learning, Neural Networks, Multilayer Perceptron Network, Feature Selection, Genetic Algorithms.

Lista de Figuras

1 Um modelo conceitual para planejamento. Fonte: Adaptado de Ghallab, Nau e Traverso (2004). 19

2 Exemplo de estado s_0 a ser representado pelo STRIPS para problemas do domínio *Blocksworld*. 21

3 Exemplo de arquivo PDDL com a definição do domínio, contendo parte da especificação do domínio *Blocksworld*. 22

4 Exemplo de arquivo PDDL com a definição de um problema do de planejamento do domínio *Blocksworld*. 23

5 Exemplo de grafo de planejamento para o problema do domínio *Blocksworld*. 26

6 Relações consideradas *mutex*. Extraído de Weld (1999). 27

7 Definição do problema do jantar. Adaptado de Weld (1999). 28

8 Exemplo de expansão do grafo de planejamento do problema do jantar com relações de *mutex*. Adaptado de Weld (1999). 29

9 Modelo de um neurônio. Fonte: Adaptada de Haykin (2007). 34

10 Arquitetura de uma rede neural MLP com uma camada oculta. 35

11 Fluxo do sinal em um neurônio de saída j. Fonte: Adaptado de Haykin (2007). 38

12 Exemplo de cromosso utilizando o alfabeto binário. 42

13 Exemplo de cruzamento entre dois cromossomos. 42

14 Exemplo de mutação de um cromossomo, com as posições que sofreram mutação em vermelho. 43

15 Diagrama de execução de uma abordagem de GAs. 43

16 Exemplo: domínio *Blocksworld*. Fonte: Adaptado de Yoon, Fern e Givan (2008). 45

17 Exemplo: domínio *Blocksworld*. Cálculo da heurística do plano relaxado. Fonte: Adaptado de Yoon, Fern e Givan (2008). 46

18 Exemplo: domínio *Blocksworld*. Cálculo da nova heurística considerando o efeito negativo das ações. Fonte: Adaptado de Yoon, Fern e Givan (2008). 46

19 Base de dados para o estado S_3 na Figura 16. Fonte: Adaptado de Yoon, Fern e Givan (2008). 49

20 Exemplo de expansão de características: primeira expansão de um conjunto de características selecionadas na primeira execução da etapa de seleção de características, composto por **g**sobre(?, ?), clear(?) e **d**segurando (?) para o domínio *Blocksworld*. 51

21 Exemplo de expansão de características: segunda expansão de um conjunto de características selecionadas na segunda execução da etapa de seleção de características, composto por **g**sobre(limpo(?), ?) e ¬**d**segurando(?), para o domínio *Blocksworld*. 51

22 Exemplo de Arquitetura de Rede MLP. 53

23 Esquema da Abordagem *LearnInPlanner*. 55

Lista de Tabelas

1 Resultados da etapa de aprendizado para o domínio *Blocksworld*. 58

2 Resultados da etapa de planejamento para o domínio *Blocksworld*. Em destaque o resultado em que a heurística *HPR+L* encontra um plano menor do que a heurística *HPR*. 59

3 Resultados obtidos por Yoon, Fern e Givan (2008) para o domínio *Blocksworld*. 60

4 Resultados da etapa de aprendizado para o domínio *Rovers*. 61

5 Resultados da etapa de planejamento para o domínio *Rovers*. Em destaque os problemas solucionados que possuem tamanho de plano menor para a heurística **HPR+L**. 61

6 Resultados da etapa de aprendizado para o domínio *Zenotravel*. 62

7 Resultados da etapa de planejamento para o domínio *Zenotravel*. Em destaque os problemas solucionados que possuem tamanho de plano menor para a heurística **HPR+L**. 63

8 Resultados da etapa de aprendizado para o domínio *Depots*. 64

9 Resultados da etapa de planejamento para o domínio *Depots*. 65

10 Resultados obtidos por Yoon, Fern e Givan (2008) para o domínio *Depots*. 65

11 Resultados da etapa de aprendizado para o domínio *Freecell*. 66

12 Resutados da etapa de planejamento para o domínio *Freecell*. Em destaque o problema solucionado com tamanho de plano menor para a heurística **HPR+L**. 67

13 Resultados obtidos por Yoon, Fern e Givan (2008) para o domínio *Freecell*. 67

Sumário

1 Introdução **13**

1.1 Objetivos 14

1.2 Justificativa 15

1.3 Hipóteses 16

1.4 Organização do Trabalho 16

2 Revisão Bibliográfica **17**

2.1 Planejamento Automatizado 17

2.1.1 Planejamento Clássico 20

2.1.2 Heurísticas e Grafo de Planejamento 25

2.1.3 Representação de Características Complexas Utilizando Sintaxe Taxonômica 30

2.2 Técnicas de Aprendizagem de Máquina Aplicadas a Problemas de Planejamento 32

2.2.1 Rede Neural Perceptron Multicamadas (MLP) 33

2.2.2 Algoritmos Genéticos 41

2.2.2.1 Implementação dos GAs: ideia geral 41

2.2.2.2 Algoritmos Genéticos e Seleção de Características 43

3 Abordagem *LearnInPlanner* **45**

3.1 Trabalhos Relacionados 47

3.2 Limitações da abordagem de Yoon, Fern e Givan (2006, 2008) 52

3.3 Abordagem *LearnInPlanner* 52

4 Discussão dos resultados **56**

4.1 Detalhes de implementação 56

4.2 Escolha de parâmetros do modelo de rede MLP e da abordagem de algoritmos genéticos (GA) 57

4.3 Experimentos com o domínio *Blocksworld* 58

4.4 Experimentos com o domínio *Rovers* 60

4.5 Experimentos com o domínio *Zenotravel* 62

4.6 Depots 64

4.7 Experimentos com o domínio *Freecell* 65

4.8 Considerações Finais 67

5 Conclusões e Trabalhos Futuros **69**

Referências **71**

Capítulo 1

Introdução

Automatizar tarefas de planejamento engloba representar computacionalmente o domínio do mundo em questão, as ações possíveis e seus efeitos nesse mundo e, uma vez definido o objetivo do planejamento, implica, ainda, no processo de busca por um conjunto de ações a serem tomadas para que esse objetivo seja alcançado (GHALLAB; NAU; TRAVERSO, 2004). Uma vantagem em automatizar o planejamento reside na utilização do grande potencial de processamento dos computadores para resolver tarefas complexas (MCDERMOTT; HENDLER, 1995).

O resultado do planejamento automatizado clássico deve ser um plano, composto por uma sequência de ações, que leva de um estado inicial qualquer a um estado (ou conjunto de estados) meta (GHALLAB; NAU; TRAVERSO, 2004). Para tal, a área de pesquisa em planejamento automatizado está interessada em algoritmos que sejam, ao mesmo tempo, genéricos o suficiente para serem aplicados a diferentes domínios de interesse e capazes de utilizar conhecimento específico de cada domínio para guiar o processo de busca por um plano (WILKINS, 1984).

Entre os algoritmos existentes para planejamento clássico, destacam-se aqueles que utilizam busca progressiva (YOON; FERN; GIVAN, 2008), que resolvem um problema de planejamento partindo do estado inicial e caminhando em direção ao estado meta (GHALLAB; NAU; TRAVERSO, 2004). Um dos motivos que justifica o sucesso desses algoritmos é o desenvolvimento de heurísticas independentes de domínio que guiam o algoritmo de busca de forma eficiente para diversos domínios (YOON; FERN; GIVAN, 2008).

Contudo, existe uma variedade de domínios em que essas heurísticas não são boas o suficiente e falham em guiar o algoritmo de busca para o planejamento. Assim, inspirando-se em Yoon, Fern e Givan (2008), o presente trabalho investigou técnicas de aprendizado de máquina com o objetivo de estimar uma nova função heurística para problemas de planejamento para domínios específicos. Como base, foi utilizada a heurística do planejador FF[1] (HOFFMANN, 2001) e de forma análoga a (YOON; FERN; GIVAN, 2008), utilizou-se algoritmos de seleção de características e aprendizado para agregar novas informações à

[1]Do original, em inglês, *Fast-Forward*.

heurística existente que auxiliem no processo de busca.

A principal ideia em que se baseia essa nova heurística apóia-se na construção de um novo espaço de características geradas a partir de informações ignoradas pela heurística do planejador FF. A esse espaço de características foi aplicada uma abordagem de algoritmos genéticos (GAs[2]) para selecionar um conjunto de características com o intuito de melhorar a função heurística de forma significativa. Uma vez selecionadas, essas características foram utilizadas para estimar a nova função heurística por meio de um processo de regressão não linear, que define o peso que cada característica exerce sobre a eficiência da heurística em guiar o algoritmo de busca.

Para realizar a regressão não linear escolheu-se a rede neural Perceptron Multicamadas (MLP[3]), que pertence ao grupo das técnicas conexionistas de aprendizado supervisionado e é capaz de realizar tarefas tanto de classificação quanto de regressão (HAYKIN, 2007).

Dessa maneira, o objetivo deste trabalho foi explorar o desempenho da utilização de modelos de rede MLP, treinados com as características selecionadas, na tarefa de estimar uma nova heurística que guiará o algoritmo de busca responsável pelo planejamento. Para avaliar o desempenho da heurística aqui proposta, realizou-se uma análise empírica utilizando domínios das Competições Internacionais de Planejamento (IPC[4]).

1.1 Objetivos

O objetivo principal do presente trabalho foi avaliar a utilização de uma rede neural MLP para estimar uma nova função que agregue à heurística do planejador FF as informações dos efeitos negativos das ações, atualmente ignoradas, e que possam melhorar a qualidade do plano solução de problemas de planejamento automatizado clássico quando o algoritmo de busca utilizado não garante encontrar a solução ótima. Para isso, os seguintes objetivos específicos foram definidos:

- Implementar um algoritmo utilizando rede neural MLP para aproximação de uma função que agregue à heurística do plano relaxado informações adicionais relevantes para guiar o algoritmo de busca na solução de problemas de planejamento de um domínio específico;
- Implementar uma abordagem de GA para seleção de características, que extraia

[2]Do original, em inglês, *Genetic Algorithms*.
[3]Do original, em inglês, *Multilayer Perceptron*.
[4]Do original, em inglês, *International Planning Competition*.

de um conjunto de problemas de planejamento solucionados de um determinado domínio informações relevantes para construir uma função heurística capaz de resolver outros problemas mais complexos desse mesmo domínio;

- Avaliar o desempenho do modelo de aprendizado para solução de problemas de planejamento do domínio específico aprendido.

1.2 Justificativa

O planejamento automatizado não é uma tarefa fácil e engloba desde a necessidade em representar o problema a ser solucionado até algoritmos eficientes e genéricos o suficiente para serem aplicados a diferentes domínios (WILKINS, 1984). Dentre as abordagens de planejamento disponíveis na literatura, escolheu-se estudar, no presente trabalho, o planejamento clássico. Nesse contexto, a aplicação de funções heurísticas para guiar os algoritmos de busca para planejamento mostrou-se uma abordagem bastante eficiente para diversos domínios de planejamento.

Nesse trabalho utilizou-se como base a heurística do planejador FF, que consiste no relaxamento do problema a ser solucionado, ignorando, para isso, o efeito negativos das ações, para realizar uma estimativa do custo em atingir o estado meta a partir de um dado estado (HOFFMANN, 2001). Entretanto, essa heurística possui dificuldade em solucionar problemas de domínios de planejamento considerados mais complexos (YOON; FERN; GIVAN, 2008).

Dessa forma, existe a necessidade de desenvolver mecanimos capazes de adicionar informações do domínio específico à heurística, de forma a melhorar sua precisão. Com a melhoria da precisão da heurística, neste trabalho, o objetivo foi melhorar a qualidade do plano encontrado por meio de um algoritmo de busca local, que não garante que a solução encontrada seja ótima. Porém, se o algoritmo de busca utilizado for ótimo, a melhoria na precisão da heurística pode melhorar a eficiência do planejador em encontrar a solução.

Nesse contexto, para auxiliar na tarefa de planejamento, este trabalho utilizou uma rede neural MLP para estimar uma nova função que agregue à heurística do FF as informações dos efeitos negativos das ações, atualmente ignoradas. A rede MLP, nesse caso, funciona como um regressor não linear e foi escolhida por ser considerada uma técnica robusta para a aproximação de funções (BATES; WATTS, 2007).

As informações agregadas à heurística são escolhidas por meio de um processo de

seleção de características que utiliza uma abordagem de GA. O GA foi escolhido por fornecer uma alternativa mais robusta para lidar com problemas que possuem um grande número de características que influenciam umas nas outras, quando comparado a abordagem gulosa (VAFAIE; IMAM, 1994) utilizada em (YOON; FERN; GIVAN, 2008).

Por meio do processo de aprendizado e seleção de características desenvolvido é possível inserir na heurística do planejador FF parcialmente as informações ignoradas completamente pela heurística atual, que resultem em estimativas de valores de custo mais preciso para atingir o estado meta a partir de qualquer estado no grafo de busca.

1.3 Hipóteses

Os experimentos realizados neste trabalho tiveram como objetivo testar as seguintes hipóteses:

1. A rede neural MLP, por meio de um processo de regressão não linear, agrega à função heurística informações adicionais que afetarão positivamente o desempenho do planejador;
2. A utilização do algorimo genético para seleção de características obterá melhores resultados que o método guloso utilizado por Yoon, Fern e Givan (2008).

1.4 Organização do Trabalho

A presente dissertação está organizada da seguinte forma. O Capítulo 2 contém uma revisão bibliográfica das técnicas utilizadas na implementação do planejador *LearnInPlanner*. O Capítulo 3 apresenta os trabalhos relacionados e detalhes da abordagem desenvolvida no presente trabalho. O Capítulo 4 discute os resultados obtidos nos experimentos para a avaliação da abordagem desenvolvida. Por fim, o Capítulo 5 apresenta as conclusões e trabalhos futuros.

Capítulo 2

Revisão Bibliográfica

Este capítulo apresenta os principais conceitos das abordagens consideradas no desenvolvimento do presente trabalho. Na Seção 2.1 são discutidos alguns conceitos importantes na área de planejamento automatizado com foco no planejamento clássico. Os conceitos básicos de planejamento clássico apresentados foram baseados principalmente em Ghallab, Nau e Traverso (2004). Na Seção 2.2 são apresentadas as técnicas de aprendizagem de máquina utilizadas neste trabalho. Dessa maneira, na Seção 2.2.1 é apresentada a rede neural Perceptron Multicamadas (MLP)[1]. Os conceitos de rede neural MLP discutidos foram baseados principalmente em Haykin (2007). Por fim, na Seção 2.2.2, são discutidos os algoritmos genéticos (GAs) e sua utilização para seleção de características.

2.1 Planejamento Automatizado

As pessoas fazem um monte de planos, às vezes para si mesmas, às vezes para outras pessoas, às vezes para máquinas. A questão que define essa seção é: Como podemos automatizar o processo de planejamento? A questão é interessante por diversas razões. Primeiro, a automação do planejamento pode dar indício sobre como as pessoas e os animais planejam seu comportamento. Em segundo lugar, os problemas complexos de planejamento podem ser resolvidos melhor com a ajuda de computadores (MCDERMOTT; HENDLER, 1995).

O desenvolvimento de técnicas para planejamento automatizado não é trivial. Automatizar o planejamento para um programa computacional implica, primeiramente, na necessidade de representar o domínio do mundo em questão, as ações possíveis e seus efeitos nesse mundo. Exige-se ainda uma compreensão do efeito de diferentes sequências de ações para solução de determinado problema (WILKINS, 1984). Nesse contexto, a sequência de ações que leva de um estado inicial qualquer a um estado meta é denominado plano.

Sistemas de planejamento automatizado podem atuar em diversos domínios, como, por exemplo, planejamento de rotas de viagens e controle de braço robótico em um am-

[1]Do original, em inglês, *Multilayer Perceptron*.

biente dinâmico. Dessa maneira, a área de pesquisa em planejamento automatizado está interessada em planejadores independentes do domínio que priorizam técnicas de planejamento aplicáveis a diferentes domínios e provêm capacidade de planejamento genérico (WILKINS, 1984). Para guiar o algoritmo de planejamento, o planejador independente do domínio deve receber como entrada a especificação do problema e conhecimento do domínio em questão descrito numa linguagem padrão.

A maioria das abordagens de planejamento utiliza conceitos de sistemas de transição de estados para a modelagem conceitual dos principais elementos do problema. Um sistema de transição de estados é descrito como $\Sigma = (S, A, E, \gamma)$, em que:

- $S = \{s_1, s_2, \ldots\}$ é um conjunto de estados;
- $A = \{a_1, a_2, \ldots\}$ é um conjunto de ações;
- $E = \{e_1, e_2, \ldots\}$ é um conjunto de eventos; e
- $\gamma : S \times A \times E \rightarrow 2^S$ é uma função de transição de estados.

A diferença entre ações e eventos é a existência ou não de controle do planejador sobre eles. Ações são transições controladas pelo plano executor, enquanto eventos são transições que correspondem à dinâmica interna do sistema. Eventos devem ser levados em conta pelo planejamento, mas não podem ser escolhidos ou disparados pelo planejador. A função de transição de estados γ define quais ações são aplicáveis a um estado s e, no caso dos eventos, quais podem *possivelmente* ocorrer quando o sistema Σ está em s.

Dado um sistema de transição de estados Σ, a tarefa do planejador é sintetizar uma sequência de ações aplicáveis a cada estado no caminho entre o estado inicial e o estado meta definidos. Um sistema de planejamento é composto por um sistema de transição de estados Σ, um planejador e um controlador, conforme o modelo conceitual mostrado na Figura 1.

O papel do planejador é gerar um plano para o controlador de acordo com a descrição de Σ, o estado inicial e o estado meta. Por sua vez, o controlador é responsável por determinar qual a próxima ação a a ser executada dependendo do estado s recebido como entrada. Como o objetivo deste trabalho é melhorar o algoritmo para geração do plano, o foco da discussão será no módulo do planejador.

Em Ghallab, Nau e Traverso (2004) é apresentado um conjunto de restrições para os sistemas de transição de estados, que originaram o Planejamento Clássico:

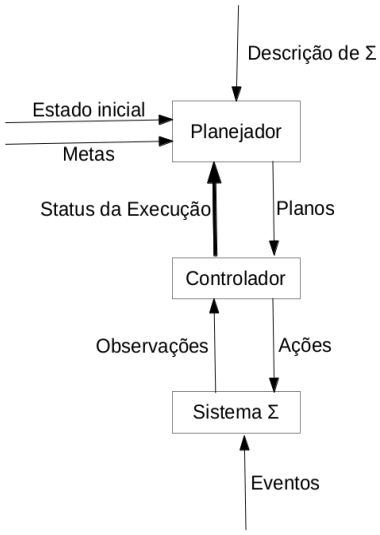

Figura 1 *– Um modelo conceitual para planejamento. Fonte: Adaptado de Ghallab, Nau e Traverso (2004).*

1. O sistema Σ possui um conjunto finito de estados;
2. O sistema Σ é considerado totalmente observável, o que significa que o planejador tem conhecimento de todos os estados de Σ;
3. O sistema Σ é determinístico. Quando uma ação ou um evento é aplicado a um estado, o resultado será um outro estado único com 100% de probabilidade;
4. O sistema é estático, permanecendo no mesmo estado até o controlador aplicar alguma ação. Isso implica que o conjunto de eventos E é vazio;
5. O planejador trabalha apenas com metas restritas, que podem ser um estado meta s_g ou um conjunto de estados meta S_g. Estados a serem evitados, restrições nas trajetórias de estado e funções de utilidade não são considerados sob esta restrição;
6. O plano solução é uma sequência finita, linear e ordenada de ações;
7. Ações não têm duração. As transições de estados são instantâneas;
8. *Offline Planning*: O planejador não está preocupado com mudanças que podem ocorrer em Σ enquanto o planejamento é realizado.

Antes de apresentar o planejamento clássico com mais detalhes, é importante ter em mente que, embora estas restrições resultem em um modelo irrealista, esse modelo

simplificado tornou possível o desenvolvimento de algoritmos e técnicas razoavelmente escalonáveis, como o planejamento neo-clássico, que, entre outras técnicas, engloba o grafo de planejamento e as heurísticas para planejamento clássico (ver Seção 2.1.2).

2.1.1 Planejamento Clássico

Conforme discutido anteriormente, o planejamento clássico assume que o sistema é determínistico, estático, finito, e totalmente observável, com metas restritas e tempo implícito. Essas restrições podem fazer o planejamento parecer trivial, uma vez que este passa a consistir em uma busca para um caminho em um grafo. Entretanto, mesmo para um domínio de aplicação simples, esse grafo pode ser extremamente grande, tornando impraticável especificá-lo explicitamente.

Dessa forma, a representação de problema de planejamento clássico não pode enumerar explicitamente os estados e transições, mas deve facilitá-la durante a execução do planejador. Em (GHALLAB; NAU; TRAVERSO, 2004) são apresentadas três formas de representação: representação baseada em teoria de conjuntos, representação clássica e representação baseada em variáveis de estado. Para os sistemas de planejamento clássico, a representação mais utilizada é a clássica, também conhecida como STRIPS[2] (FIKES; NILSSON, 1971), e, portanto, será a utilizada neste trabalho.

A representação STRIPS foi concebida utilizando notações derivadas de lógica de primeira ordem. Uma linguagem de primeira ordem é constituída por um conjunto de constantes, funções e símbolos de predicado, em que cada função e símbolo de predicado são associados à uma aridade (número de argumentos) (MCALLESTER; GIVAN, 1993). O STRIPS utiliza uma representação de ações com três entidades (FIKES; NILSSON, 1971): modelo de mundo inicial, ações e estado meta. O modelo de mundo inicial é representado por um conjunto de fatos que descreve o estado atual do mundo. Assim, dada uma ação e um estado, com STRIPS, é possível determinar se a ação pode ser realizada no estado e qual o estado resultante após aplicar a ação.

Para um problema do domínio *Blocksworld*, em que o objetivo é dispor um conjunto de blocos em uma determinada configuração, o estado apresentado na Figura 2 seria definido pelos fatos: $segurando(A)$, que significa que a garra está segurando o bloco A; $sobre(B, table)$, que significa que o bloco B está sobre a mesa; $sobre(C, table)$, que define que o bloco C está sobre a mesa; $limpo(B)$, que significa que nenhum bloco está em cima

[2]Do original, em inglês, *STanford Research Institute Problem Solver*.

do bloco B; e $limpo(C)$, que define que nenhum bloco está em cima do bloco C.

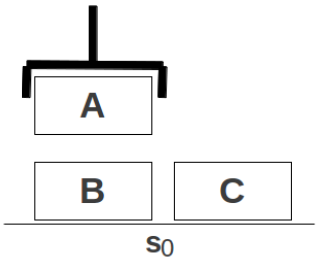

Figura 2 *– Exemplo de estado s_0 a ser representado pelo STRIPS para problemas do domínio Blocksworld.*

Por sua vez, as ações são um conjunto de operadores, com a descrição dos efeitos da ação e das precondições necessárias para que a ação seja aplicável a um estado. Os efeitos das ações são divididos em dois grupos (FIKES; NILSSON, 1971):

- Efeitos positivos ou *add list*: uma lista de relações que tornam-se verdadeiras após aplicar a ação. Por exemplo, ao aplicar a ação $empilhar(A, B)$ ao estado s_0 na Figura 2, a lista dos efeitos positivos será composta por $sobre(A, B)$, $limpo(A)$ e $garravazia()$;
- Efeitos negativos ou *delete list*: uma lista de relações que tornam-se falsas após aplicar a ação. Para o mesmo exemplo, ao aplicar a ação $empilhar(A, B)$, a lista dos efeitos negativos conterá $segurando(A)$ e $limpo(B)$.

A representação STRIPS assume que as relações ignoradas na descrição da ação permanecem inalteradas mesmo após a aplicação da ação. Por fim, a terceira entidade representada pelo STRIPS é o estado meta, que é definido de forma análoga ao modelo de mundo inicial e descreve como deve ser o modelo de mundo após a aplicação do plano solução (FIKES; NILSSON, 1971). Para descrever a representação STRIPS, neste trabalho, foi utilizada a linguagem PDDL[3]. Os componentes de uma tarefa de planejamento descrito na linguage PDDL são (MCDERMOTT et al., 1998):

- Objetos: representa os itens que compõe o mundo de interesse;
- Predicados: representa as propriedades dos objetos definidos e pode ser verdadeiro ou falso;

[3]Do original, em inglês, *Planning Domain Definition Language*.

- Estado inicial: representa o estado do mundo no qual o planejamento será iniciado;
- Meta: contém um conjunto de fatos que devem ser verdadeiros ao final do planejamento;
- Ações/Operadores: representa as formas pelas quais os estados do mundo podem ser alterados.

As tarefas de planejadores especificados por meio da linguagem PDDL devem ser descritas em dois tipos de arquivos. O primeiro arquivo deve conter a definição do domínio, com os predicados e ações disponíveis para o mundo modelado (MCDERMOTT et al., 1998). A Figura 3 apresenta parte da especificação do domínio *Blocksworld* por meio da linguagem PDDL,

```
(:types bloco)
(:predicates (sobre ?x - bloco ?y - bloco)
        (sobreamesa ?x - bloco)
        (limpo ?x - bloco)
        (garravazia)
        (segurando ?x - bloco)
        )

(:action pegar
      :parameters (?x - bloco)
      :precondition (and (limpo ?x) (sobreamesa ?x) (garravazia))
      :effect
      (and (not (sobreamesa ?x))
        (not (limpo ?x))
        (not (garravazia))
        (segurando ?x)))
```

Figura 3 – *Exemplo de arquivo PDDL com a definição do domínio, contendo parte da especificação do domínio Blocksworld.*

em que (MCDERMOTT et al., 1998):

- *:types* define quais os tipos de objetos que o domínio é composto. No exemplo, o mundo é composto por blocos;
- *:predicates* define os predicados aplicáveis aos objetos;
- *:action* define cada ação possível para o domínio, sendo que podem ser definidas quantas ações forem necessárias. Cada ação é composta por parâmetros, pré-condições que precisam ser satisfeitas para que seja possível realizar a ação e efeitos

que representa as alterações nos estados causadas pela aplicação da ação. No exemplo foi definida apenas uma ação denominada *pegar*, que representa a tarefa de pegar um bloco por meio de uma garra.

O outro arquivo descrito por meio da linguagem PDDL consiste na definição do problema a ser solucionado, com os objetos que compõem o mundo modelado, o estado inicial considerado e qual a meta a ser alcançada (MCDERMOTT et al., 1998). A Figura 4 mostra um exemplo de problema do domínio *Blocksworld* descrito em linguagem PDDL,

```
(:domain blocos)
(:objects d b a c - bloco)
(:init (limpo c) (limpo a) (limpo b) (limpo d) (sobreamesa c)
(sobreamesa a) (sobreamesa b) (sobreamesa d) (garravazia))
(:goal (and (sobre d c) (sobre c b) (sobre b a)))
)
```

Figura 4 *– Exemplo de arquivo PDDL com a definição de um problema do de planejamento do domínio Blocksworld.*

em que (MCDERMOTT et al., 1998):

- *:objects* define os objetos considerados pelo problema. No exemplo, são considerados quatro blocos denominados como d, b, a e c;
- *:init* é composto pelos predicados que descreve o estado inicial;
- *:goal* define os predicados que devem ser verdadeiros ao atingir o estado meta.

Uma vez definida a representação para o domínio, é necessário escolher o algoritmo de planejamento para a geração dos planos. No planejamento clássico, os algoritmos podem atuar em um espaço de busca constituído por um subconjunto do espaço de estados, em que cada nó corresponde a um estado do mundo, cada arco corresponde a uma transição de estado, e o plano atual corresponde ao caminho atual no espaço de busca.

Os algoritmos de planejamento que realizam busca no espaço de estados podem fazer Busca Progressiva[4] ou Busca Regressiva[5]. A busca regressiva inicia-se a partir do estado meta e, por meio da aplicação dos operadores de planejamento, são produzidas sub-metas. O objetivo é chegar a um conjunto de sub-metas que são satisfeitas no estado inicial.

[4]Do original, em inglês, *Forward Search.*
[5]Do original, em inglês, *Backward Search.*

Por sua vez, a busca progressiva resolve um problema de planejamento partindo do estado inicial e caminhando em direção ao estado meta. Assim, dado um estado atual s_i, é verificado se o mesmo é ou não o estado meta. Caso não seja, são identificadas todas as ações aplicáveis ao estado e, uma vez escolhida a ação executada, é computado o estado sucessor resultante da aplicação da ação. Esse processo é repetido até que o estado meta seja atingido ou que não exista nenhuma ação aplicável ao estado atual, caso em que o planejamento falha. Essa abordagem de busca, segundo Yoon, Fern e Givan (2008), é ao mesmo tempo uma das mais antigas técnicas disponíveis para planejamento e também uma das mais populares.

Os algoritmos baseados em busca progressiva possuem um desempenho que os coloca entre os melhores do estado da arte, incluindo o planejamento clássico. Um dos principais motivos para o sucesso da busca progressiva é o surgimento de poderosas heurísticas para planejadores independentes do domínio que apresentam bom desempenho em muitos domínios de planejamento (YOON; FERN; GIVAN, 2008). Essas heurísticas podem ser admissíveis ou não. Uma heurística é considerada admissível se para cada estado no espaço de busca, o valor da heurística nunca ultrapassa o custo verdadeiro de alcançar o estado objetivo a partir do estado considerado. Isso significa que uma heurística admissível nunca superestima o custo de alcançar o objetivo, sendo, portanto, uma heurística otimista (RUSSELL; NORVIG, 2002).

Entretanto, existem domínios em que heurísticas, mesmo sendo admissíveis, não funcionam bem, resultando em fracasso no planejamento. Isto motiva a investigação de técnicas de Aprendizado de Máquina que podem extrair do domínio específico o conhecimento necessário para melhorar os algoritmos de busca progressiva no espaço de estados (YOON; FERN; GIVAN, 2008).

Uma outra preocupação ao se planejar por meio de algoritmos de busca é a completude. Para garantir que um algoritmo de busca progressiva seja completo é preciso que todas as ações aplicáveis a um estado sejam consideradas. A ordem que os estados serão considerados vai depender da estratégia de busca adotada: busca em largura, busca em profundidade, busca gulosa, entre outras (VIDAL, 2004).

2.1.2 Heurísticas e Grafo de Planejamento

O planejamento como um *framework* de busca heurística iniciou-se com os planejadores ASP[6] (BONET; LOERINCS; GEFFNER, 1997), HSP[7] (BONET; GEFFNER, 1998), HSPr[8] (BONET; GEFFNER, 1999, 2000) e HSP2[9] (BONET; GEFFNER, 2001). Os planejadores ASP e HSP utilizam uma função heurística derivada da representação STRIPS para o problema de planejamento. Dado o problema original P, essa heurística consiste em gerar um problema de planejamento 'relaxado' P' que ignora todos os efeitos negativos das ações para obter uma estimativa do tamanho do plano dado partindo de um estado qualquer até o estado meta (BONET; GEFFNER, 1998).

A heurística desenvolvida por Bonet, Loerincs e Geffner (1997) consiste em construir um plano relaxado para P' e calcular o valor do custo para atingir o estado meta por meio deste plano. Este valor servirá como uma estimativa do custo para atingir o estado meta no problema original P (BONET; GEFFNER, 1998). Por ignorar os efeitos negativos, o plano relaxado não é necessariamente um plano válido para o estado s. Essa estimativa é utilizada para guiar algoritmos de busca como busca A*, busca gulosa e busca de subida de encosta[10](BONET; GEFFNER, 1998). Inspirado principalmente no HSP, Hoffmann (2001) introduziu o algoritmo FF, que utiliza a mesma função heurística do HSP, porém, com a aplicação do método GRAPHPLAN como base para estimar o tamanho do plano por meio de um grafo de planejamento.

O GRAPHPLAN é um sistema de planejamento desenvolvido por Blum e Furst (1997). Eles introduziram a técnica de geração de planos baseada na construção e análise do grafo de planejamento, que superou o desempenho das técnicas que fazem a busca diretamente no espaço de estados (HOFFMANN; NEBEL, 2001). O algoritmo do GRAPHPLAN constrói primeiramente a estrutura compacta do grafo de planejamento, organizando as informações de forma a minimizar a complexidade de busca necessária. Essa estrutura possui complexidade, em termos de tempo e tamanho, polinomial (BLUM; FURST, 1997).

O grafo de planejamento é um grafo direcionado e em níveis com três tipos de arestas e dois tipos de nós. Os níveis do grafo de planejamento se alternam entre nível de predicados, contendo os nós de predicados que representam os estados, e nível de ações, com os nós de ações aplicáveis aos predicados (estados) do nível anterior. As arestas, por sua vez,

[6]Do original, em inglês, *Action Selection for Planning.*
[7]Do original, em inglês, *Heuristic Search Planner.*
[8]Do original, em inglês, *Heuristic Regression Planning.*
[9]Do original, em inglês, *Heuristic Search Planner 2.0.*
[10]Do original, em inglês, *Hill-Climbing Search.*

representam a relação entre ações e predicados e podem ser dos seguintes tipos (BLUM; FURST, 1997):

- arestas de precondição: conecta as ações do nível i às suas precondições no nível i;
- arestas de adição: conecta as ações do nível i aos seus efeitos positivos no nível de predicados $i + 1$;
- arestas de exclusão: conecta as ações do nível i aos seus efeitos negativos no nível de predicados $i + 1$.

Um fator que faz com que a construção do grafo de planejamento seja rápida é a não-exigência de independência entre as ações. Isso significa que podem existir ações pertencentes a um mesmo nível que interferem umas nas outras: caso em que uma ação remove um efeito positivo ou negativo de outra. Além disso, uma propriedade do grafo de planejamento é que, dados os t níveis de ação criados, caso exista um plano válido usando t ou menos passos, então esse plano será um subgrafo do grafo de planejamento (BLUM; FURST, 1997).

A Figura 5 mostra um exemplo de grafo de planejamento para o problema do domínio *Blocksworld*. As arestas de adição são representadas pela linha contínua, as arestas de exclusão são representadas pela linha pontilhada e os predicados que se mantém são ligados pelas linhas com um ponto à esquerda.

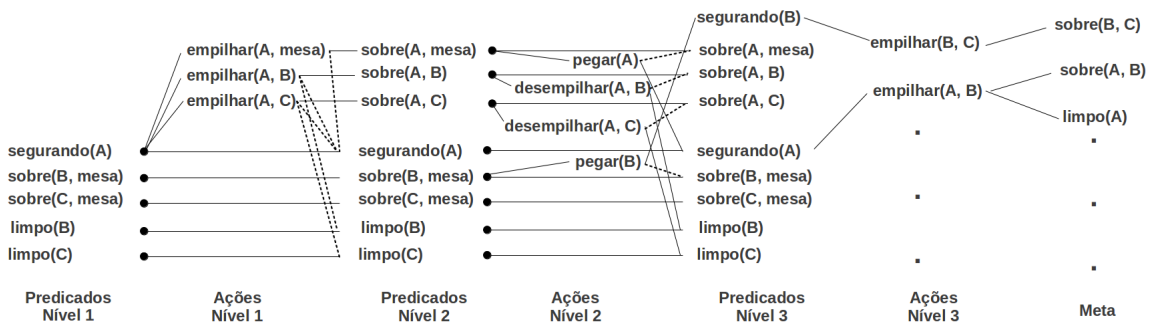

Figura 5 *– Exemplo de grafo de planejamento para o problema do domínio Blocksworld.*

O planejamento utilizando o GRAPHPLAN inicia com a construção do primeiro nível, que conterá o nível de predicados com os fatos de estado inicial e o nível de ações aplicáveis ao estado inicial. O grafo é estendido nos passos posteriores, nível por nível, até atingir

um que contenha todas as sub-metas, sem que haja contradições entre elas (HOFFMANN; NEBEL, 2001).

O grafo de planejamento construído representa as ações de maneira paralela em cada nível de ação. Isso implica que os k níves de ação podem representar um plano com mais de k ações. Contudo, esse paralelismo das ações nos níveis não significa que as ações poderão ser executadas ao mesmo tempo. Segundo Weld (1999), a inferência no que diz respeito a relações binárias de exclusão mútua (*mutex*) entre os nós de um mesmo nível é peça chave para a eficiência do GRAPHPLAN.

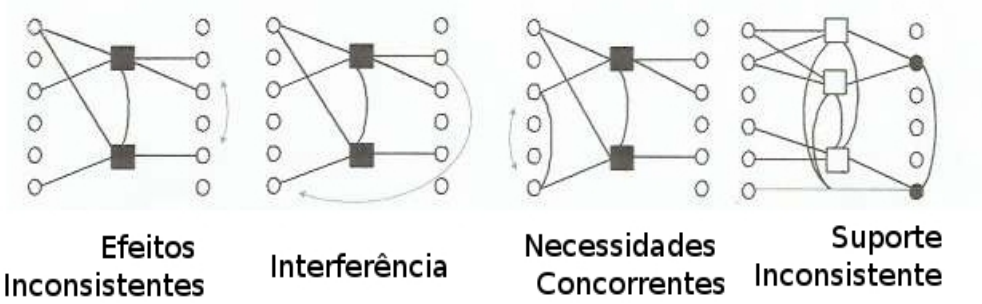

Figura 6 – *Relações consideradas mutex. Extraído de Weld (1999).*

Relações *Mutex*, ilustradas na Figura 6, devem ser consideradas tanto no nível das ações quanto no nível dos predicados. No nível das ações, duas ações são consideradas *mutex* em três situações (WELD, 1999):

1. Efeitos inconsistentes, que ocorre quando o efeito de uma ação é a negação do efeito de outra ação;
2. Interferência, que ocorre quando uma ação deleta a precondição de outra ação;
3. Necessidades concorrentes, que ocorre quando as ações têm precondições mutuamente exclusivas no nível $i - 1$.

No nível dos predicados, dois predicados são considerados *mutex* em uma de duas situações (WELD, 1999):

1. Quando um predicado é a negação de outro predicado;
2. Suporte inconsistente, que ocorre quando todas as maneiras de atingir um predicado por meio das ações no nível $i - 1$ são *mutex* aos pares.

Para exemplificar, considere o problema de preparar um jantar especial, apresentado em Weld (1999) para alguém, que nesse momento está dormindo na sua casa, conforme definido na Figura 7, e que a meta é jogar o lixo fora, preparar o jantar e embrulhar o presente. Considere ainda que no estado inicial apenas os predicados *mãosLimpas* (representando que as mãos estão limpas), *lixo* (definindo que a casa tem lixo para jogar fora) e *silêncio* (silêncio necessário para que a pessoa alvo da surpresa não acorde) são verdadeiras.

```
Condições iniciais: (e (lixo) (mãosLimpas) (silêncio))
Meta: (e (jantar) (presente) (não (lixo)))
Ações:
    cozinhar :pré-condição (mãosLimpas)
             :efeito (jantar)
    embrulhar   :pré-condição (silêncio)
             :efeito (presente)
    carregar :pré-condição
             :efeito (e (não (lixo)) (não (mãosLimpas)))
    arrumar  :pré-condição
             :efeito (e (não (lixo)) (não (silêncio)))
```

Figura 7 *– Definição do problema do jantar. Adaptado de Weld (1999).*

A Figura 8 mostra a primeira expansão possível do problema. Nesse exemplo, a ação *carregar*, que tem como efeito negar o predicado *lixo*, com a permanência do estado *lixo*, leva a uma situação de *mutex* de efeitos inconsistentes. Por sua vez, a ação *arrumar* nega o estado *silêncio* que é precondição para a ação *embrulhar*, colocando essas ações em situação de *mutex* de interferência. Um suporte inconsistente ocorre entre ¬*silêncio* no nível de predicados 2 e *silêncio* no nível atual. Tem-se ainda os predicados no nível 2 *lixo*, *mãosLimpas* e *silêncio* com seus respectivos *mutex* de negação ¬*lixo*, ¬*mãosLimpas* e ¬*silêncio*. Para o problema não existe situação de necessidades concorrentes, uma vez que nenhuma das ações definidas possuem precondições mutuamente exclusivas.

A expansão do grafo de planejamento ocorre até atingir um nível em que todos os predicados do estado meta estejam presentes sem a ocorrência de *mutex* entre si. Essa condição de parada é necessária, porém não suficiente, para existência de um plano (WELD, 1999). Uma vez que o algoritmo de expansão atingiu o critério de parada, uma busca regressiva é realizada para procurar um plano válido (HOFFMANN; NEBEL, 2001). Para extrair a solução, o algoritmo de busca considera os caminhos que não contenham *mutex* entre ações ou predicados (WELD, 1999).

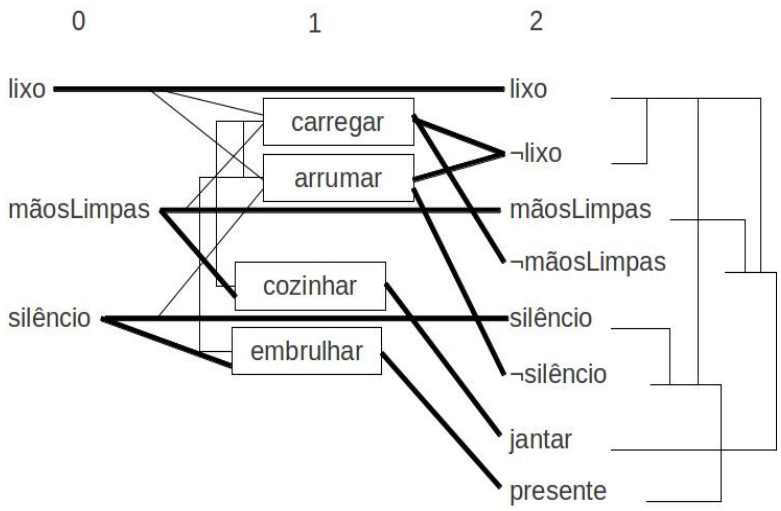

Figura 8 – *Exemplo de expansão do grafo de planejamento do problema do jantar com relações de mutex. Adaptado de Weld (1999).*

Um plano válido é encontrado quando o algoritmo atinge o nível zero do grafo e todos os predicados do estado inicial estão presentes e não são *mutex*. No caso do algoritmo não encontrar nenhum caminho válido no grafo, o grafo de planejamento é novamente expandido até atingir um outro nível em que o critério de parada da expansão é satisfeito e o algoritmo de busca tenta novamente extrair um plano válido.

Segundo Weld (1999), a diferença entre o ponto de parada de uma primeira expansão e o ponto de parada da segunda é que neste o número de relações *mutex* tende a ser menor. Além disso, cada predicado do último nível da segunda expansão tende a ter mais ações de suporte, o que implica em mais caminhos possíveis para o algoritmo de busca regressiva explorar. Esses dois fatores contribuem para um aumento na chance de existência de um plano válido.

Para calcular a heurística do planejador FF, Hoffmann (2001) introduziu uma versão relaxada do GRAPHPLAN, na qual o grafo é construído considerando apenas o efeito positivo das ações. O resultado do planejamento com o grafo relaxado será um plano relaxado $\langle O_0, \ldots, O_{m-1} \rangle$, onde cada O_i é o conjunto de ações selecionado em paralelo no passo i e m é o número do primeiro nível contendo todos os estados meta. A heurística é então estimada como sendo $h(S) := \sum_0^{m-1} |O_i|$, em que $|\cdot|$ é o operador de cardinalidade (HOFFMANN, 2001; HOFFMANN; NEBEL, 2001).

Outra diferença com relação ao planejador HSP é o emprego de uma técnica de poda

para selecionar os sucessores para cada nó de busca mais promissores e outra técnica para podar as ramificações que levam a algum estado meta prematuramente. Por fim, Hoffmann e Nebel (2001) propuseram uma alteração no algoritmo de busca de subida de encosta, denominada *enforced form*, que combina busca local e sistemática para o planejamento. A discussão mais detalhada dessas alterações estão fora do escopo deste trabalho.

A última consideração a se fazer sobre a função heurística é quanto a sua limitação. Embora existam domínios em que a heurística apresentada obtém ótimos resultados, existem outros vários domínios em que a mesma leva a falhas no planejamento, por subestimar demais o tamanho real do plano. Isto oferece a oportunidade para que novos algoritmos ou abordagens melhorem essas heurísticas para os domínios em que esta apresenta dificuldades em guiar a busca (YOON; FERN; GIVAN, 2008).

Em Yoon, Fern e Givan (2008) foi proposta uma nova abordagem que visa a aumentar a precisão da heurística por meio da inclusão de informação de novas características do problema de planejamento a ser solucionado. As características utilizadas para agregar informação à heurística do planejamento foram representadas por meio da sintaxe taxônomica apresentada a seguir.

2.1.3 Representação de Características Complexas Utilizando Sintaxe Taxonômica

A sintaxe taxonômica aqui apresentada foi proposta por McAllester e Givan (1993) para trabalhar com lógica de primeira ordem e regras de inferências poderosas e computacionalmente tratáveis. Assim, a sintaxe taxonômica é uma linguagem de primeira ordem, útil para descrever uma expressão de classe C e relações taxonômicas entre as expressões de classe (MCALLESTER; GIVAN, 1993).

Uma linguagem de primeira ordem, por sua vez, é constituída por um conjunto de constantes, funções e símbolos de predicado, em que cada função e símbolo de predicado são associados à uma aridade (número de argumentos). De forma análoga aos termos de primeira ordem, uma expressão de classe é construída a partir de variáveis, constantes, símbolos de funções e símbolos de predicados. Entretanto, enquanto os termos de primeira ordem denotam indivíduos, as expressões de classe denotam conjuntos. Assim, uma expressão de classe pode ser (MCALLESTER; GIVAN, 1993):

- uma variável;

- um símbolo de constante;
- um símbolo de predicado;
- uma aplicação $f(C_1, \ldots, C_n)$ de um símbolo de função n-ária f a n expressões de classe;
- uma aplicação $R(C_1, \ldots, C_{n-1})$ de um símbolo de relação n-ária R (com $n > 1$) a $n - 1$ expressões de classe.

Para exemplificar o uso de funções ou relações que têm como argumentos outras expressões de classe, considere que um símbolo de predicado **A-CHILD** é uma expressão de classe que representa um conjunto qualquer de crianças e que um símbolo de função **A-MOTHER** é uma expressão de classe que representa um conjunto de mães. Dessa forma, a expressão de classe **A-MOTHER(A-CHILD)** determina o conjunto de mães de algumas crianças. Com essas declarações atômicas sobre as relações taxonômicas entre expressões de classe podem ser construídas fórmulas de sintaxe taxonômica. Uma fórmula taxonômica pode assumir um dos seguintes tipos (MCALLESTER; GIVAN, 1993):

- uma fórmula de classificação, **Is(C, W)**, sendo que C e W são expressões de classe;
- uma fórmula de existência, **Ex(C)**, onde C é uma expressão de classe;
- uma fórmula de determinação, **Det(C)**, sendo C uma expressão de classe;
- uma fórmula de intersecção, **Int(C, W)**, onde C e W são expressões de classe;
- uma combinação Booleana de fórmulas taxonômicas;
- ou uma fórmula quantificada da forma $\forall x \Phi(x)$ ou $\exists x \Phi(x)$, sendo que $\Phi(x)$ é uma fórmula taxonômica.

Intuitivamente, a semântica das fórmulas taxonômicas corresponde simplesmente a ler as fórmulas como se fossem sentenças em inglês (MCALLESTER; GIVAN, 1993). Por exemplo, a fórmula taxonômica **Is(x, AMOTHEROF(y))** é verdadeira apenas quando o par $\langle x, y \rangle$ está contido na relação denotada por **AMOTHEROF**. Por sua vez, a fórmula **Is(x, AMOTHER)** é verdadeira apenas quando o valor da variável x é um elemento do conjunto denotado pela expressão de classe **AMOTHER**.

Dados os predicados P, o conjunto de expressões de classe possíveis são dados por:

$$C := \textbf{a-thing} | P_1 | C \cap C | \neg C | (P\ C_1 \ldots C_{i-1}\ ?\ C_{i+1} \ldots C_{n(p)}), \tag{2.1}$$

em que C e C_j são expressões de classe, P_1 é um predicado de aridade 1 qualquer, e P é qualquer símbolo de predicado de aridade 2 ou maior. As expressões de classe primitivas são constituídas pelo conjunto de todos os objetos, denotado por **a-thing**, e os conjuntos de objetos para os quais os predicados P são verdadeiros, representados pelos predicados de aridade 1. Por meio de operações de complementação, intersecção ou composição relacional é possível obter as expressões de classe compostas (YOON; FERN; GIVAN, 2006, 2008).

Dessa forma, dada uma expressão de classe C, a sua profundidade $depth(C)$ é definida como: **0** para **a-thing** e predicados de aridade 1; $depth(C_1 \cap C_2) = 1 + max(depth(C_1), depth(C_2))$; $depth(\neg C) = 1 + depth(C)$; e a profundidade da expressão $(P \;\; C_1 \ldots C_{i-1} \;\; ? \;\; C_{i+1} \ldots C_{n(P)})$ é $1 + max(depth(C_1), \ldots, depth(C_{n(P)}))$. De acordo com essa definição, o número de expressões de classe pode ser infinito. Para o processo de aprendizado é, portanto, necessário colocar um limite na profundidade permitida (YOON; FERN; GIVAN, 2008).

Com relação à semântica, dada uma expressão de classe C e uma base de dados D, $C[D]$ representa o conjunto de objetos para a expressão de classe C com respeito à base D. O conjunto de tuplas de objetos para o qual o predicado P é verdadeiro é denotado por $P[D]$. Por exemplo, para o problema do domínio *Blocksworld*, **a-thing** e $sobreamesa[D]$ representa o conjunto de todos os blocos e o conjunto dos blocos que estão sobre a mesa, respectivamente, na base de dados D (YOON; FERN; GIVAN, 2008).

No caso de $C = C_1 \cap C_2$, $C[D] = C_1[D] \cap C_2[D]$. Por exemplo, $(limpo \cap sobreamesa)[D]$ denota o conjunto de blocos 'limpos' (sem outros por cima) e que estão sobre a mesa. Se $C = \neg C'$, então, $C[D] = \textbf{a-thing} - C'[D]$. E por fim, se $C = (P \;\; C_1 \ldots C_{i-1} \;\; ? \;\; C_{i+1} \ldots C_{n(p)})$, então $C[D]$ é o conjunto de todas as constantes c para as quais existe $c_j \in C_j[D]$ e a tupla $(c_1, \ldots, c_{i-1}, c, c_{i+1}, \ldots, c_{n(R)})$ está em $P[D]$. Para ficar mais claro, pode-se considerar novamente o problema do domínio *Blocksworld*, em que $(sobre\ limpo\ ?)[D]$ é o conjunto de todos os blocos que estão imediatamente abaixo de algum bloco 'limpo' (YOON; FERN; GIVAN, 2008).

2.2 Técnicas de Aprendizagem de Máquina Aplicadas a Problemas de Planejamento

No presente trabalho, a inclusão de novas características foi realizada por meio da inclusão parcial da informação dos efeitos negativos das ações, que na heurística do plano

relaxado eram totalmente ignoradas. Para isso, foi utilizada uma abordagem de algoritmo genético (GA) com o objetivo de selecionar as características mais relevantes no processo de estimativa do tamanho do plano. Para ponderar cada característica selecionada foi utilizado um modelo de rede neural MLP.

2.2.1 Rede Neural Perceptron Multicamadas (MLP)

A motivação principal da pesquisa em redes neurais é o reconhecimento do cérebro humano como um sistema de processamento de informações, que age de forma diferente do computador digial convencional. Nesse sentido, o cérebro pode ser visto como um computador altamente complexo, não linear, paralelo e organizado de forma a realizar certos processamentos, como reconhecimento de padrões, percepção e controle motor, de forma bem mais rápida que os computadores digitais.

As redes neurais artificiais, usualmente denominadas apenas redes neurais, foram desenvolvidas como generalizações dos modelos matemáticos da cognição humana ou biologia neural, assumindo-se que (FAUSETT, 1994):

1. A informação é processada por vários elementos simples chamados neurônios;
2. Os sinais são transferidos entre os neurônios por meio de conexões;
3. Cada conexão tem um valor de peso associado, que, na rede neural típica, pondera o sinal transmitido;
4. Cada neurônio aplica uma função de ativação (geralmente não linear) as suas entradas para determinar o seu sinal de saída.

Segundo Haykin (2007), um neurônio possui três elementos básicos (ilustrado na Figura 9):

1. Sinapses (ou elos de conexão): cada sinapse possui um valor associado, denominado peso sináptico. Durante o processamento, o peso sináptico w_{kj} é multiplicado pelo sinal x_j na entrada do neurônio j conectado ao neurônio k;
2. Somador $\sum$: os sinais de entrada de um neurônio k devem ser somados, ponderados pelos pesos das respectivas sinapses do neurônio. Essa operação constitui uma combinação linear;

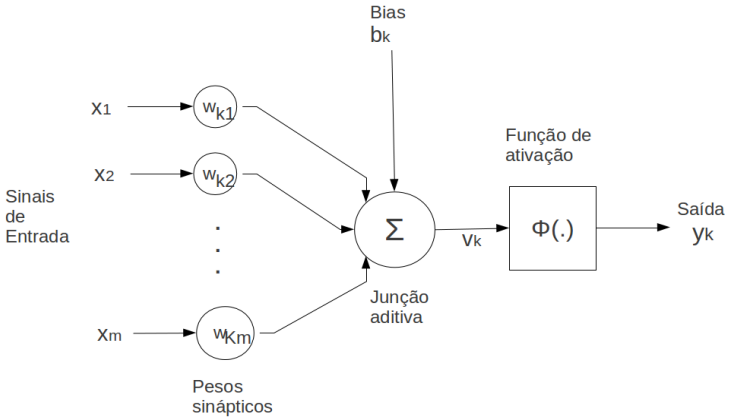

Figura 9 *– Modelo de um neurônio. Fonte: Adaptada de Haykin (2007).*

3. Função de ativação $\Phi(.)$: a amplitude da saída de um neurônio é restrita, pela função de ativação, geralmente a um intervalo unitário fechado $[0, 1]$ ou alternativamente $[-1, 1]$.

Segundo Rumelhart, Hintont e Williams (1986), neurônios podem receber sinais por meio da inclusão de uma entrada extra que sempre terá o valor 1 ou -1. O peso dessa entrada extra é chamado *bias* e é equivalente a um limiar de sinal oposto. Esse peso sináptico pode ser tratado de forma igual a qualquer outro peso.

De forma análoga ao cérebro humano, a capacidade de processamento da rede neural é possível devido à habilidade para adquirir conhecimento por meio do que se denomina experiência. Essa experiência é acumulada com o tempo conforme as interações com o ambiente. A esse processo dá-se o nome de aprendizagem e o procedimento utilizado para o processo de aprendizagem é chamado de algoritmo de aprendizagem ou algoritmo de aprendizado.

Existem três paradigmas principais de aprendizado em redes neurais: supervisionado, não supervisionado e por reforço. O aprendizado supervisionado é análogo ao aprendizado com um professor, em que juntamente com o dado (ou vetor) de treinamento é informada para a rede a resposta desejada e a mesma é utilizada para ajustar o modelo aprendido. No aprendizado não-supervionado não existe um professor externo para fornecer uma resposta esperada, a rede deve ser capaz de se auto-organizar para aprender a representação dos dados. Por fim, no aprendizado por reforço é fornecido apenas um estímulo positivo ou negativo para informar quando a rede acertou ou errou (HAYKIN, 2007).

A rede neural Perceptron Multicamadas (MLP)[11] pertence ao grupo das redes neurais com aprendizado supervisionado. A arquitetura da rede MLP é composta por camadas de neurônios, sendo uma camada de entrada, uma ou mais camadas ocultas e uma camada de saída. A rede MLP é totalmente conectada: um neurônio em qualquer camada está conectado a todos os neurônios da camada anterior. A adição de camadas ocultas oferece a habilidade para descobrir representações intermediárias úteis para o aprendizado. Segundo Mitchell (1997), essa habilidade provê um grau de flexibilidade importante que permite ao modelo inferir novas características implicitas a partir das características fornecidas como entrada da rede. Além disso, quanto mais camadas são usadas na rede neural, mais características complexas podem ser inferidas.

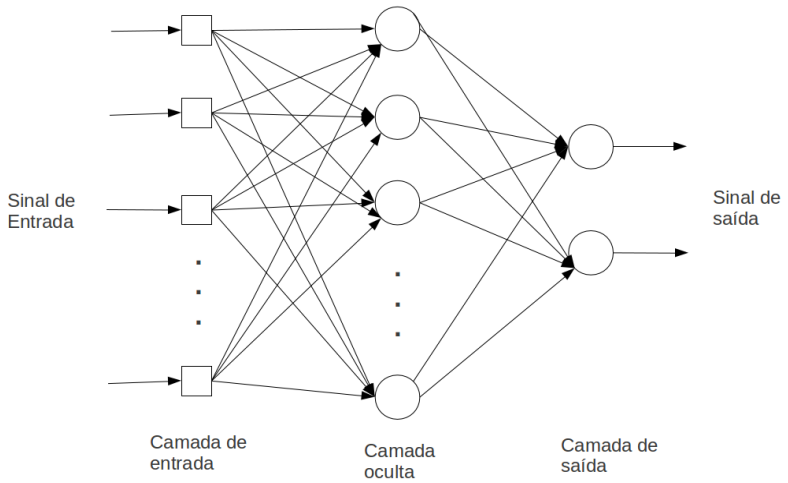

Figura 10 *– Arquitetura de uma rede neural MLP com uma camada oculta.*

A arquitetura da rede MLP é apresentada na Figura 10. O sinal de entrada que a rede recebe como estímulo do ambiente é propagado para frente, da esquerda para direita, e de camada em camada, sem sinais de *feedback*. Isto a coloca na família das redes neurais *feed-forward* (BISHOP, 1996). A camada de saída fornece ao meio ambiente os sinais de saída que representam a resposta da rede para a tarefa designada, que pode ser de classificação ou regressão.

O benefício da adição de camadas ocultas é aumentar o espaço de hipóteses que uma rede neural pode representar. Segundo Russell e Norvig (2002), com uma única, mas suficientemente ampla, camada oculta é possível representar qualquer função contínua e com duas camadas é possível representar mesmo funções não-contínuas. Entretanto, não existe uma forma de conhecimento prévio que defina (ou guie) qual tipo de representação

[11]Do original, em inglês, *Multilayer Perceptron*.

é adequada para determinado problema. O modelo de rede neural MLP é escolhido com base em experimentos específicos para a tarefa em questão (RUSSELL; NORVIG, 2002).

A comparação de diferentes topologias de redes neurais para aproximar uma dada função pode ser feita considerando fatores como tamanho, profundidade ou peso. Essas medidas são definidas, respectivamente, pelo número de neurônios, o tamanho do maior caminho entre um neurônio de entrada e um neurônio de saída, e a soma total dos valores absolutos de todos os pesos da rede (ORPONEN, 1994).

Uma vez que um conjunto de treinamento, contendo os dados observados e as respostas desejadas, está disponível e a função de ativação para os neurônios está definida, o desafio do algoritmo de treinamento é encontrar um conjunto de pesos sinápticos que garanta que, para cada sinal de entrada, a rede produza um sinal de saída igual (ou próximo) ao desejado (RUMELHART; HINTONT; WILLIAMS, 1986).

Quando os neurônios da camada de entrada estão diretamente conectados aos neurônios da camada de saída é relativamente fácil definir regras de aprendizado que ajustem os pesos de forma a reduzir progressivamente a diferença entre a saída obtida e a saída desejada. Ao introduzir unidades ocultas, o aprendizado torna-se ao mesmo tempo mais interessante e mais difícil. Neste caso, o procedimento de aprendizado deve ser capaz de definir em quais circunstâncias os neurônios ocultos devem ser ativados ou não (RUMELHART; HINTONT; WILLIAMS, 1986). A dificuldade aqui encontra-se na não-existência de um valor desejado de saída para os neurônios da camada oculta (BISHOP, 1996).

Para resolver esse impasse, foi desenvolvido o algoritmo *Backpropagation*. Pressupondo que as funções de ativação da rede são diferenciáveis, as ativações dos neurônios de saída passam a ser funções diferenciáveis das variáveis de entrada, dos pesos e dos *biases*. Assim, basta definir uma função de erro que também será uma função diferenciável dos pesos e avaliar as derivadas do erro com respeito aos pesos para encontrar um valor de peso que minimize essa função (BISHOP, 1996).

Antes de iniciar a discussão sobre o funcionamento do algoritmo *Backpropagation*, é importante notar que o termo *Backpropagation* pode ser utilizado para referenciar a uma variedade de contextos na literatura sobre redes neurais artificiais. Por exemplo, a arquitetura da rede MLP algumas vezes é chamadada de rede neural *Backpropagation*, independente do algoritmo utilizado para avaliar as derivadas das funções e da função de erro escolhida (BISHOP, 1996). Neste trabalho, o termo *Backpropagation* será empregado para descrever o treinamento da rede utilizando o algoritmo do gradiente descendente aplicado à função soma dos erros quadráticos. A derivação do algoritmo *Backpropagation*

apresentada nesta seção tem como base a demonstração realizada em Haykin (2007).

O treinamento utilizando o algoritmo *Backpropagation* é dividido em dois passos. O primeiro é o passo para frente, em que é fornecido o n-ésimo exemplo de entrada para a rede e os sinais funcionais são propagados para frente, da camada de entrada para a de saída. Nesse caso, a rede computará um valor de saída $y_j(n)$ baseado no conhecimento que esta possui armazenado no momento. O segundo passo, para trás, utilizará o valor de saída para calcular o erro e reajustar o valor dos pesos com base no mesmo.

Nesse segundo passo, o sinal de erro de cada neurônio j da camada de saída, para o n-ésimo exemplo de treinamento, deve ser calculado conforme:

$$e_j(n) = d_j(n) - y_j(n), \tag{2.2}$$

em que $d_j(n)$ é a saída esperada e $y_j(n)$ é a saída observada para o neurônio j. O valor instantâneo da energia do erro para o neurônio j será igual a $\frac{1}{2}{e_j}^2(n)$. O valor do erro total é obtido pela soma dos termos $\frac{1}{2}{e_j}^2(n)$ de todos os neurônios da camada de saída, representado por:

$$\epsilon(n) = \frac{1}{2}\sum_{j \in C} {e_j}^2(n), \tag{2.3}$$

em que C é o conjunto de todos os neurônios da camada de saída da rede. Dado o conjunto de treinamento com tamanho igual a N, a função de erro quadrado médio é obtida somando-se todos os termos $\epsilon(n)$ para todos os padrões contidos no conjunto, normalizando-se em relação ao tamanho do conjunto N, conforme a equação a seguir:

$$\epsilon_{\text{med}} = \frac{1}{N}\sum_{n=1}^{N} \epsilon(n). \tag{2.4}$$

Dessa maneira, a medida de desempenho de aprendizado de um modelo de rede MLP é representada pela função custo ϵ_{med} e o objetivo do processo de aprendizagem é ajustar os parâmetros livres da rede para minimizar ϵ_{med}. Os ajustes dos pesos são realizados de acordo com os respectivos erros calculados para cada padrão apresentado à rede durante o treinamento. Isto significa que na prática o valor a ser minimizado será $\epsilon(n)$, sendo que a média aritmética dessas correções representa uma estimativa da alteração real que resultaria da modificação dos pesos utilizando ϵ_{med}.

A Figura 11 ilustra o cálculo do sinal funcional para um neurônio j. O valor $v_j(n)$ representa o somatório dos sinais de saída dos neurônios da camada anterior (ou sinais de entrada, no caso da camada de entrada) ponderados pelos pesos, calculados por $v_j(n) = \sum_{i=0}^{m} w_{ji}(n)y_i(n)$. O valor m representa o número total de entradas (excluindo o *bias*) aplicados ao neurônio j; w_{ji} corresponde ao peso sináptico entre o neurônio j e o neurônio i da camada anterior e $y_i(n)$ é o sinal de saída do neurônio i. O sinal $y_j(n)$ é calculado pela aplicação da função de ativação $f_j(.)$ sobre $v_j(n)$, resultando em $y_j(n) = f_j(v_j(n))$.

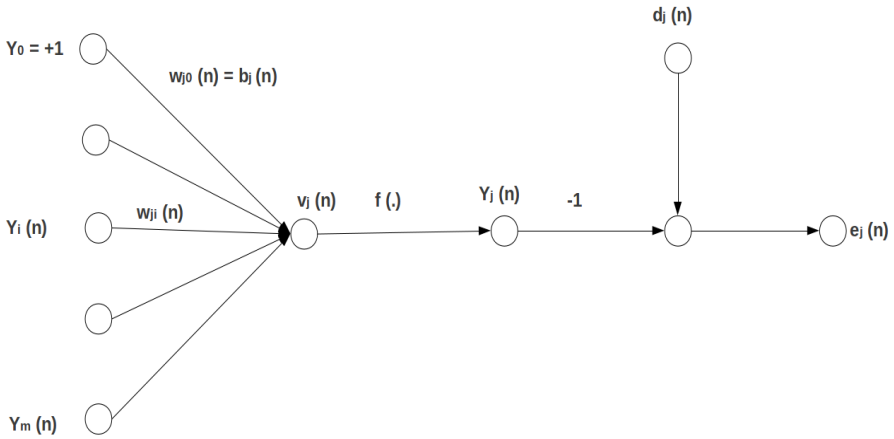

Figura 11 *– Fluxo do sinal em um neurônio de saída j. Fonte: Adaptado de Haykin (2007).*

A correção $\Delta w_{ji}(n)$ aplicada pelo algoritmo *Backpropagation* ao peso sináptico $w_{ji}(n)$ é proporcional à derivada parcial $\frac{\partial \epsilon(n)}{\partial w_{ji}(n)}$, que é igual ao gradiente expresso por:

$$\frac{\partial \epsilon(n)}{\partial w_{ji}(n)} = -e_j(n)f'_j(v_j(n))y_j(n). \tag{2.5}$$

Uma vez que o valor $y_j(n)$ na Equação 2.5 corresponde a uma das entradas do neurônio, o gradiente local pode ser definido como $\delta_j(n) = -e_j(n)f'_j(v_j(n))$. Como o objetivo é buscar uma alteração nos pesos que minimize $\epsilon(n)$, deve ser aplicado o método de descida do gradiente no espaço de pesos, por meio do gradiente negativo. Dessa forma, tem-se:

$$\Delta w_{ji}(n) = -\eta \frac{\partial \epsilon(n)}{\partial w_{ji}(n)}, \tag{2.6}$$

$$\Delta w_{ji}(n) = \eta \delta_j(n) y_j(n), \tag{2.7}$$

em que η é o parâmetro da taxa de aprendizado do algoritmo *Backpropagation*. Para os neurônios da camada de saída, o sinal de erro associado pode ser calculado diretamente

e, portanto, o gradiente local é definido conforme a Equação 2.8:

$$\delta_j(n) = e_j(n) f'_j(v_j(n)). \tag{2.8}$$

No caso dos neurônios da camada oculta, não existe uma respota desejada especificada. O sinal de erro para um neurônio oculto é calculado, portanto, em termos dos sinais de erro de todos os neurônios aos quais está diretamente conectado pela direita, de acordo com a Equação 2.9:

$$\delta_j(n) = f'_j(v_j(n)) \sum_k \delta_k(n) w_{kj}(n), \tag{2.9}$$

em que j é o neurônio oculto e k são os neurônios da camada diretamente conectada ao mesmo pela direita. O fator $f'_j(v_j(n))$ das Equações 2.8 e 2.9 depende unicamente da função de ativação associada ao neurônio j. A função de ativação a ser utilizada neste trabalho para os neurônios da MLP será a sigmóide logística, definida como:

$$f_j(v_j(n)) = \frac{1}{1 + exp(-v_j(n))} \tag{2.10}$$

e derivada igual a:

$$f'_j(v_j(n)) = f_j(v_j(n))[1 - f_j(v_j(n))]. \tag{2.11}$$

O algoritmo *Backpropagation* fornece uma aproximação para a trajetória no espaço de pesos calculada pelo método da descida mais íngreme. Assim, a taxa de aprendizado define a intensidade das variações dos pesos sinápticos da rede durante o aprendizado. Se o valor da taxa de aprendizado for pequeno, haverá menos variações nos valores dos pesos e será obtida uma trajetória mais suave, entretanto, com aprendizado mais lento. Por outro lado, se a taxa de aprendizado for muito grande, as grandes variações nos pesos sinápticos podem tornar a rede instável.

Para aumentar a velocidade do processo de aprendizado, sem correr o perigo da instabilidade, foi criado um termo denominado *momentum*, que representa uma modificação pequena na atualização dos pesos a partir do fator de erro calculado na iteração anterior. Nesse caso, o valor de $\Delta w_{ji}(n)$ é modificado para:

$$\Delta w_{ji}(n) = \alpha \Delta w_{ji}(n-1) + \eta \delta_j(n) y_i(n), \tag{2.12}$$

em que α é uma constante que pondera o termo *momentum*. Um outro benefício do termo *momentum* é evitar que o processo de aprendizado termine em um mínimo local raso na superfície do erro, uma vez que algoritmos de otimização, como o gradiente descendente, possuem critério de treinamento não convexo. O Algoritmo 1 mostra o pseudocódigo do algoritmo *Backpropagation* apresentado em (RUSSELL; NORVIG, 2002).

```
1  Backpropagation(exemplos, MLP)
   //exemplos compreende um conjunto de exemplos, em que cada um é
     representado por um vetor de entrada x e um vetor de saída y, que
     correspondem às características de um padrão de dado e a resposta
     desejada, respectivamente.
   //MLP compreende uma rede MLP com L camadas, todos os pesos w_{j,i} que
     conectam os neurônios e função de ativação f.  η é a taxa de
     aprendizado do modelo.
   //Neste algoritmo não é utilizado o termo momentum
2  repeat
3  foreach e in exemplos do
       //Passo para frente
4      foreach neurônio j na camada de entrada do
5          a_j ⟵ x_j[e]
6      for l = 2 to L do
7          in_i ⟵ Σ_j w_{j,i} a_j
8          a_j ⟵ f(in_i)
       //Passo para trás
9      foreach neurônio i na camada de saída do
10         Δ_i ⟵ f'(in_i) × (y_i[e] − a_i)
11     for l = L − 1 to 1 do
12         foreach neurônio j na camada l do
13             Δ_j ⟵ f'(in_j) Σ_i w_{j,i} Δ_i
14             foreach neurônio j na camada l + 1 do
15                 w_{j,i} ⟵ w_{j,i} + η × a_j × Δ_i
16 until Algum critério de parada ser satisfeito
17 return MLP
```

Algorithm 1: Algoritmo *Backpropagation*.

2.2.2 Algoritmos Genéticos

A inspiração para o desenvolvimento dos Algoritmos Genéticos (GAs) advém do mecanismo de modelos genéticos do sistema natural (VAFAIE; IMAM, 1994). GAs consistem em técnicas de busca adaptativas, introduzidas inicialmente por Holland (1975), e em um modelo de mudanças genéticas em uma população de indivíduos com três elementos básicos (JONG, 1988):

1. Uma noção Darwiniana de adaptação, que define o quanto um indivíduo influencia nas gerações futuras;
2. Um operador de reprodução, que gera os indivíduos para a próxima geração;
3. Os operadores genéticos, que definem os genes herdados pelos indivíduos da próxima geração, com base no material genético dos seus pais.

Um GA é um sistema adaptativo que, ao invés de necessitar de informação prévia sobre o problema a ser resolvido (JONG, 1980), define um processo de adaptação que mantém uma população de estruturas a partir das quais são derivadas novas estruturas (JONG, 1988) por meio de técnicas inspiradas no sistema evolutivo natural, como hereditariedade, seleção natural, cruzamento e recombinação (HOLLAND, 1975). Assim, a informação sobre o problema é dinamicamente acumulada para gerar uma solução aceitável (JONG, 1980).

Na Seção 2.2.2.1 é descrito, de forma geral, o funcionamento dos GAs. Na Seção 2.2.2.2 é discutido brevemente os benefícios da utilização de abordagens de GAs, com foco em seleção de características, de acordo com o trabalho de Vafaie e Imam (1994).

2.2.2.1 Implementação dos GAs: ideia geral

A implementação de uma abordagem de GAs inicia-se pela geração, geralmente de forma aleatória, de uma população inicial (BACK; FOGEL; MICHALEWICZ, 2000), em que cada indivíduo representa uma solução no espaço de soluções possíveis para o problema em questão (JONG, 1980). Os indivíduos são identificados unicamente por uma *string*, denominada cromossomo, gerada a partir de algum alfabeto, por exemplo o alfabeto binário $\{0, 1\}$, e que guarda o material genético em posições específicas da string (genes) (JONG, 1980). A Figura 12 contém um exemplo de cromossomo utilizando o alfabeto binário para representação.

1	0	1	1	0	0	1	0

Figura 12 – *Exemplo de cromosso utilizando o alfabeto binário.*

Na segunda etapa, a população inicial passa pelo processo de seleção, no qual cada indivído é avaliado por uma função aptidão de modo que os melhores tenham mais changes de serem escolhidos para participar do processo de reprodução. A reprodução pode incluir tanto a cópia exata de alguns indivíduos para a próxima geração quanto o 'nascimento' de um novo indivíduo por meio do cruzamento de dois outros indivíduos (BACK; FOGEL; MICHALEWICZ, 2000; JONG, 1980). Para o segundo caso, são utilizados operadores genéticos, sendo os mais conhecidos o cruzamento e a mutação (JONG, 1980).

Dados dois indivíduos escolhidos da população atual para o cruzamento, deve ser selecionado o ponto de cruzamento. A parte inicial do primeiro indivíduo é, então, concatenada com a parte final do segundo indivíduo, gerando os indivíduos 'filhos' (JONG, 1980). A Figura 13 ilustra o processo de cruzamento entre dois indivíduos, em que o cruzamento ocorre entre as posições 2 e 3 dos cromossomos 'pais'. Formas de cruzamento mais elaboradas podem ser encontradas em (MITCHELL, 1997).

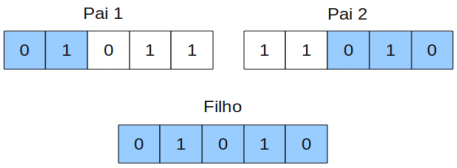

Figura 13 – *Exemplo de cruzamento entre dois cromossomos.*

Segundo Jong (1980), uma limitação desse processo é que o espaço de busca do algoritmo fica restrito ao subespaço representado pela população atual. Se, por exemplo, todos os indivíduos da população, representados pelo alfabeto binário, contêm apenas um zero na primeira posição do cromossomo, o cruzamento nunca produzirá um novo indivíduo com um 1 nesta posição.

Assim, o operador de mutação foi criado para suprir essa limitação. A mutação gera um novo indivíduo modificando de forma independente e aleatória um ou mais genes de um cromossomo qualquer. Embora este operador não seja considerado um operador primário, pois a probabilidade de um gene sofrer mutação pode ser menor que 0.001 (tanto no sistema natural quanto no artificial), o mesmo garante que a probabilidade de,

por exemplo, um zero ocorrer na primeira posição do gene nunca seja nula (JONG, 1980). Um exemplo de mutação é apresentado na Figura 14.

Figura 14 – *Exemplo de mutação de um cromossomo, com as posições que sofreram mutação em vermelho.*

A população da próxima geração será composta, portanto, pelos indivíduos copiados da população inicial, pelos novos indivíduos gerados no cruzamento e pelos indivíduos mutados. Os processos anteriores são, então, repetidos até que a solução desejada seja encontrada ou algum outro critério de parada seja atingido (JONG, 1980). A Figura 15 ilustra os passos da execução das abordagens de GA.

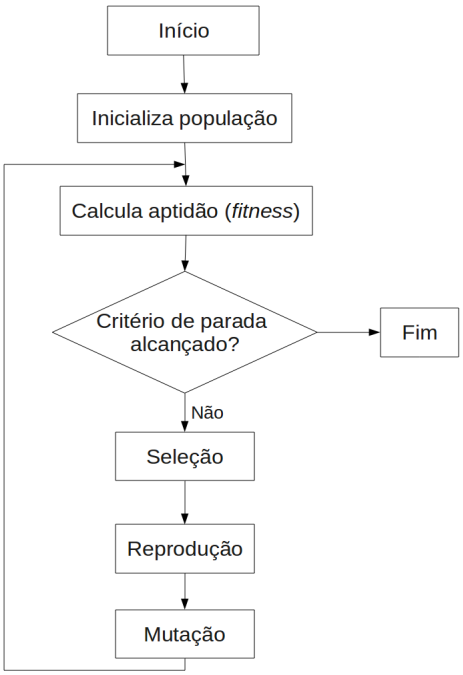

Figura 15 – *Diagrama de execução de uma abordagem de GAs.*

2.2.2.2 Algoritmos Genéticos e Seleção de Características

O interesse na utilização de GAs para tarefas de aprendizado é principalmente devido ao fato que estes provêm um conjunto de heurísticas de busca independente de domínio que possuem desempenho significamente superior a vários outros métodos de aprendizagem

tradicionais. Segundo Jong (1988), existem evidências consideráreis de que GAs são úteis para problemas de otimização de funções e problemas NP-complexos.

A possibilidade de aplicação de GA, utilizada neste trabalho, é a seleção de características que possam representar, de forma compacta e satisfatória, o problema em questão. A principal meta das técnicas de seleção de características é definir um subconjunto de características que seja pequeno de modo a diminuir a complexidade do sistema, mas que não seja tão pequeno a ponto de levar à perda de desempenho em termos de precisão (VAFAIE; JONG, 1992).

As técnicas de seleção de características existentes na literatura seguem dois tipos de abordagens: dependentes de domínio, em que estratégias específicas são desenvolvidas baseadas em conhecimentos sobre o problema; técnicas independentes de domínio, que utilizam heurísticas genéricas, essencialmente algoritmos gulosos, para selecionar as características. A segunda abordagem é útil quando não existe conhecimento sobre o domínio disponível ou é impraticável explorá-lo.

A busca gulosa e os GAs enquadram-se na segunda abordagem e são ferramentas úteis para resolver problemas de seleção de características em que tanto o tamanho do conjunto de características quanto o desempenho do sistema que as utilizará são importantes (VAFAIE; JONG, 1992). Segundo Vafaie e Imam (1994), que realizaram um estudo comparativo entre GAs e algoritmos baseados em abordagens gulosas, a busca gulosa tem tendência a cair em picos locais causados por ruídos ou interdependência entre características.

As abordagens de GAs são mais robustas para seleção de características, entretanto, ao custo do aumento na complexidade computacional. Este trabalho utilizou uma abordagem de GA para seleção de características importantes para cálculo da funções heurísticas para planejamento, em contraposição ao trabalho desenvolvido em (YOON; FERN; GIVAN, 2008), que propôs uma abordagem gulosa para esse propósito. Detalhes de como os GAs serão utilizados neste trabalho serão apresentados no Capítulo 3.

Capítulo 3

Abordagem LearnInPlanner

Uma função heurística $H(s, A, g)$, para um dado domínio de planejamento D, deve estimar o custo de atingir o estado meta g a partir do estado s utilizando o conjunto de ações A. Em algoritmos tradicionais de planejamento clássico, sendo o exemplo mais conhecido o algoritmo FF (HOFFMANN, 2001), essa função é calculada por meio do relaxamento do problema a ser solucionado, ignorando o efeito negativo das ações. Entretanto, essa heurística pode subestimar o custo real para atingir a meta (YOON; FERN; GIVAN, 2006).

Considere, por exemplo, o problema do domínio *Blocksworld* apresentado na Figura 16. A partir do estado inicial, é possível alcançar a meta por qualquer um dos três planos relaxados, dependendo de qual será a próxima ação executada: $soltar(A)$, $empilhar(A, B)$, $empilhar(A, C)$, que resultam nos estados S_1, S_2 e S_3, respectivamente. Mediante a utilização da heurística do plano relaxado, ilustrado na Figura 17, a ação a ser executada deveria ser $empilhar(A, B)$ e, portanto, o próximo estado seria S_2, com tamanho do plano para esse caminho estimado em 3, contra 4 para S_1 e S_3.

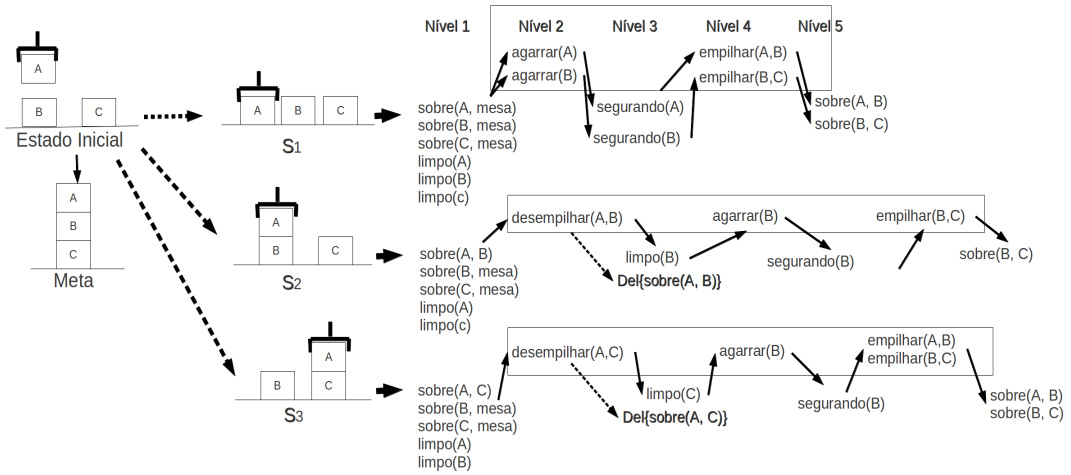

Figura 16 *– Exemplo: domínio Blocksworld. Fonte: Adaptado de Yoon, Fern e Givan (2008).*

Contudo, se forem analisados os efeitos negativos no caminho de S_2, verifica-se que

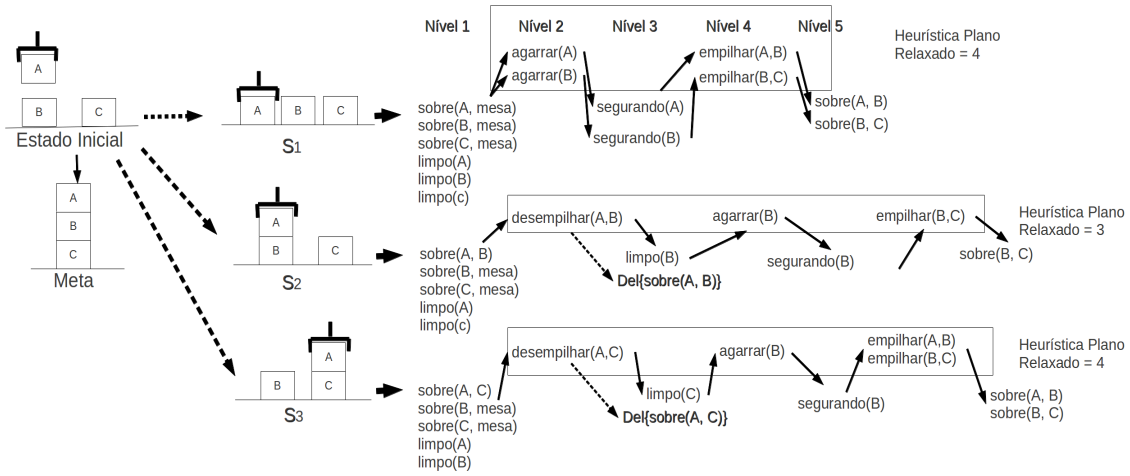

Figura 17 – *Exemplo: domínio Blocksworld. Cálculo da heurística do plano relaxado. Fonte: Adaptado de Yoon, Fern e Givan (2008).*

$sobre(A, B)$ é ao mesmo tempo efeito negativo de $desempilhar(A, B)$ e está presente na meta. Dessa forma, para melhorar a estimativa da heurística deveria-se somar ao comprimento do plano relaxado os termos de efeitos negativos. Segundo Yoon, Fern e Givan (2006) e Yoon, Fern e Givan (2008), é possível computar o número de ocorrências dos fatos '*sobres*' presentes simultaneamente no conjunto de efeitos negativos de uma ação do plano relaxado e na meta, resultando nos valores 0 para S_1 e 1 para S_2, conforme Figura 18.

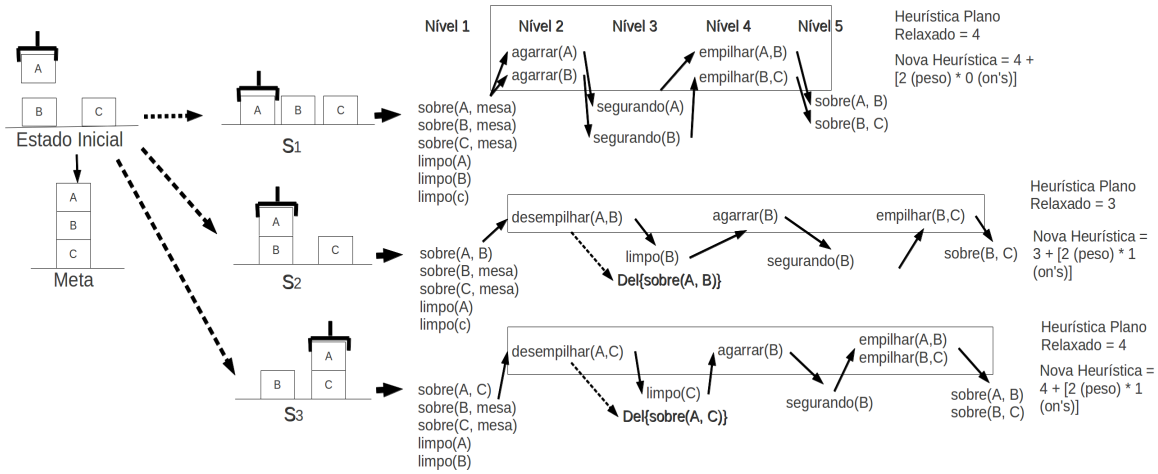

Figura 18 – *Exemplo: domínio Blocksworld. Cálculo da nova heurística considerando o efeito negativo das ações. Fonte: Adaptado de Yoon, Fern e Givan (2008).*

Supondo o peso do número de ocorrências de '*sobre*' como igual a 2 e acrescentando o

resultado da multiplicação entre o número de ocorrências de '*sobres*' e o peso do predicado '*sobre*' ao tamanho do plano relaxado, obtém-se uma nova heurística que atribui o valor 4 para S_1 e 5 para S_2. Estes novos valores possibilitam elencar os caminhos S_1 e S_2 na ordem correta, de forma que o melhor caminho seria por S_1 ao invés de S_2. Apesar de simples, esse exemplo demonstra a utilidade de características derivadas dos efeitos negativos das ações e dos *mutex*, ignorados pela heurística do plano relaxado (YOON; FERN; GIVAN, 2008).

Nos trabalhos de Yoon, Fern e Givan (2006, 2008), nos quais a presente abordagem se baseia, foram apresentadas novas ideias de como utilizar abordagens de aprendizado de máquina para incorporar parcialmente as informações ignoradas pelo plano relaxado. Na Seção 3.1, discute-se essas ideias anteriormente desenvolvidas, que resultaram numa função heurística que calcula um valor de custo mais próximo do real para atingir o estado meta. Na Seção 3.2 aponta-se algumas de suas limitações. Por fim, na Seção 3.3, apresenta-se a proposta desenvolvida no presente trabalho.

3.1 Trabalhos Relacionados

Com a finalidade de melhorar a precisão da função heurística de planos relaxados, Yoon, Fern e Givan (2006) e Yoon, Fern e Givan (2008) propuseram um algoritmo de aprendizado capaz de extrair parcialmente as informações sobre os efeitos negativos, ignorados pelo plano relaxado. A informação extraída é empregada na aproximação de uma função que compensa parcialmente a lacuna entre o valor estimado pela heurítica do plano relaxado e a distância real até a meta. A nova função heurística $H(s, A, g)$ passa a ser definida como a soma da função do plano relaxado $RPL(s, A, g)$ e da função com as informação dos efeitos negativos $\Delta(s, A, g)$.

A função $\Delta(s, A, g)$ é aproximada por meio de uma combinação linear de funções f_{Ci}, na forma $\Delta(s, A, g) = \sum_i w_i \cdot f_{Ci}(s, A, g)$, em que cada C_i representa as características a serem selecionadas para a nova estimativa. Essas características são definidas por meio de uma sintaxe taxonômica como expressões de classe (ver Seção 2.1.3) e seus valores são computados como a cardinalidade de uma certa expressão de classe sobre uma base de dados de fatos provenientes dos planos relaxados.

Para a geração dessa base de dados, é criado um conjunto de treinamento contendo exemplos de planos que solucionam problemas de planejamento de um domínio específico, juntamente com a definição do domínio em questão. Esses planos, por sua vez, são

gerados pelo algoritmo do planejador FF (HOFFMANN, 2001). Assim, dado um problema de planejamento (s_0, A, g) e a correspondente trajetória para solução $(s_0, s_1, \ldots, s_n)$, é criado um conjunto J que contém um exemplo de plano para cada estado na trajetória do problema, na forma $\{((s_i, A, g), n - i) | i = 0, \ldots, n - 1\}$. Nesse caso, $n - i$ é a distância observada entre o estado s_i e o meta g.

Para exemplificar, considere um problema $P = (s_0, A, s_4)$, em que a solução, gerado pela o algoritmo do planejador FF, é $(s_0, s_1, s_2, s_3, s_4)$, e, portanto, o valor a ser estimado pela heurística é 4. Para esse caso, o conjunto de treinamento J seria composto pelos exemplos $(s_0, A, s_4) \rightarrow 4$, $(s_1, A, s_4) \rightarrow 3$, $(s_2, A, s_4) \rightarrow 2$, e $(s_3, A, s_4) \rightarrow 1$.

Como o objetivo é estimar a diferença entre a distância observada pelo planejador e a estimada pela heurística do plano relaxado, é construído um conjunto J', derivado de J, utilizado pelo algoritmo de aprendizado, no qual a distância observada foi substituída por essa diferença. Para o exemplo anterior, considere que a distância estimada pela heurística do plano relaxado é igual a 1. O conjunto J', derivado de J, seria composto pelos exemplos $(s_0, A, s_4) \rightarrow 3$, $(s_1, A, s_4) \rightarrow 2$, $(s_2, A, s_4) \rightarrow 1$ e $(s_3, A, s_4) \rightarrow 0$.

A partir desse conjunto de treinamento é realizada a captura das informações sobre o plano relaxado em uma base de dados $D(s, A, g)$, definida inicialmente em (YOON; FERN; GIVAN, 2006) e aprimorada em (YOON; FERN; GIVAN, 2008). Dado um nó de busca qualquer (s, A, g), a base de dados deve conter as seguintes informações:

- Todos os fatos em s;
- O nome de cada ação a_i no plano relaxado a partir de s. Por exemplo, $empilhar(A, B)$ em S_2 na Figura 16;
- Para cada fato na lista de efeitos positivos de uma ação a_i no plano relaxado, adicionar o fato à base de dados com o prefixo **a**, como, por exemplo, $\mathbf{a}segurando(B)$ presente na lista de efeitos positivos da ação $agarrar(B)$ em S_2 na Figura 16;
- De forma similar, para cada fato na lista de efeitos negativos de uma ação a_i no plano relaxado, adicionar o fato à base de dados com prefixo **d**, como, por exemplo, $\mathbf{d}sobre(A, B)$ presente na lista de efeitos negativos da ação $desempilhar(A, B)$ em S_2 na Figura 16;
- Para cada fato na meta g, adicionar o prefixo **g** ao predicado do fato e adicioná-lo na base de dados, como $\mathbf{g}sobre(A, B)$ e $\mathbf{g}sobre(B, C)$, na Figura 16;

- Para cada predicado que aparece simultaneamente na meta e no estado atual, adicionar o prefixo **c**, como **c**$sobre(A, B)$ presente no estado atual de S_2 e na meta na Figura 16.

A Figura 19 mostra um exemplo dessa base de dados para o estado S_3 da Figura 16. Como pode-se perceber, é possível extrair não somente informações dos efeitos negativos, como outras informações do estado atual, da trajetória e da meta em si. O desafio passa, então, a ser a escolha das características mais representativas e que ajudem a estimar o custo de atingir a meta por meio dos vários caminhos possíveis. A escolha do método de seleção de características demanda cautela, pois, segundo Vafaie e Imam (1994), existe uma forte relação entre a natureza dos dados e o comportamento do método de seleção.

sobre(A,C), sobre(B, mesa), sobre(C, mesa), limpo(A), limpo(B), garravazia()
desempilhar(A, C), agarrar(B), empilhar(A,B), empilhar(B,C)
asegurando(A), **a**limpo(C), **a**segurando(B), **a**sobre(B,C), **a**sobre(A,B)
dsobre(A,C), **d**garravazia(), **d**garravazia(), **d**garravazia(), **d**sobre(B, mesa), **d**segurando(B), **d**segurando(A), **d**limpo(C), **d**limpo(B)
gsobre(A,B), **g**sobre(B,C)
csobre(C,mesa)

Figura 19 – *Base de dados para o estado S_3 na Figura 16. Fonte: Adaptado de Yoon, Fern e Givan (2008).*

Yoon, Fern e Givan (2006, 2008) utilizaram um método guloso para selecionar as características para um determinado problema de planejamento, a partir de um conjunto de características Φ inicial constituído pelo conjunto de expressões de classe de profundidade 0 e 1. O processo consiste em adicionar gradativamente as características ponderadas pela combinação linear e avaliar o desempenho da nova função heurística, por meio do cálculo de uma medida estatística denominada *R-squared*, até atingir uma solução satisfatória. O processo de seleção de característica é intercalado com uma etapa de expansão, em que o conjunto de características Φ' escolhidas no passo anterior são combinadas, gerando um conjunto novo de candidatas Φ.

Para isso, o conjunto Φ' é utilizado como semente para o procedimento **ExpandirCaracteristicas**. Conforme apresentado no Algoritmo 2, além do conjunto de características Φ', o novo conjunto expandido deve conter as características geradas pelas três funções de expansão consideradas em Yoon, Fern e Givan (2008) e utilizadas no Algoritmo 2:

- **ExpansaoRelacional**: dada a semente C, retorna todas as expressões da forma

```
1 ExpandirCaracteristicas(D, Φ')
2 Φ ⟵ Φ'
3 foreach C ∈ Φ do
4     Φ ⟵ Φ ∪ ExpansaoRelacional(D, C) ∪ Especializacao(D, C) ∪ {¬C}
5 return Φ
```

Algorithm 2: Função ExpandirCaracteristicas

$(P\ c_0 \ldots c_{j-1}\ c_{j+1} \ldots c_{j-1}(?)c_{j+1} \ldots c_{n(P)})$, em que P é um predicado de aridade maior que 1, c_i são todos os **a-thing**, e $i, j \leq n(P)$. O resultado representa todas as maneiras possíveis de restringir um único argumento de um predicado de C, não sendo necessário colocar nenhuma outra restrição sobre o predicado. No problema do domínio *Blocksworld*, isso gera, por exemplo, para a expressão de classe *holding*, a extensão (**g**sobre segurando ?), que descreve os blocos que, no estado meta, devem estar sob os blocos seguros pela garra mecânica no estado atual.

- **Especializacao**: gera todas as expressões de classe que podem ser criadas substituindo-se uma única sub-expressão c' de C de profundidade 0 ou 1 pela intersecção de c' e outra expressão de classe de profundidade 0 ou 1. Por exemplo, dada a expressão do domínio *Blocksworld* (sobre ? **a-thing**), que representa todos os blocos em cima de qualquer outro bloco, uma expressão de classe gerada pode ser (sobre ? (**a-thing** ∩ **g**limpo)), representando os blocos que estão em cima de qualquer outro bloco no estado atual e que devem estar 'limpos' no estado meta.

- **Negacao ($\neg C$)**: Finalmente, são adicionados os complementos de todas as classes sementes. Por exemplo, no domínio *Blocksworld*, o complemento da expressão de classe (**c**segurando ? **a-thing**), todos os blocos seguros pela garra mecânica atualmente, é ¬(**c**segurando ? **a-thing**), todos os blocos que não estão seguros pela garra.

Para exemplificar o processo de expansão, considere um conjunto de características composto pelas expressões de classe **g**sobre(?, ?), clear(?) e **d**segurando(?) para o domínio *Blocksworld* selecionadas na primeira execução da etapa de seleção de características. A Figura 20 mostra o conjunto de características obtido a partir da primeira execução da etapa de expansão de características. Uma vez que as expressões de classe utilizadas na expansão não possuem restrições sobre os parâmetros, observe, por exemplo, que aplicar a função de expansão **ExpansaoRelacional** entre as expressões de classe limpo(?) e **g**segurando(?), resultando na expressão de classe **g**segurando(limpo(?)), produz o mesmo resultado de aplicar a função de expansão **Especializacao** á expressão de

classe **g**segurando(?) com a expressão de classe limpo(?).

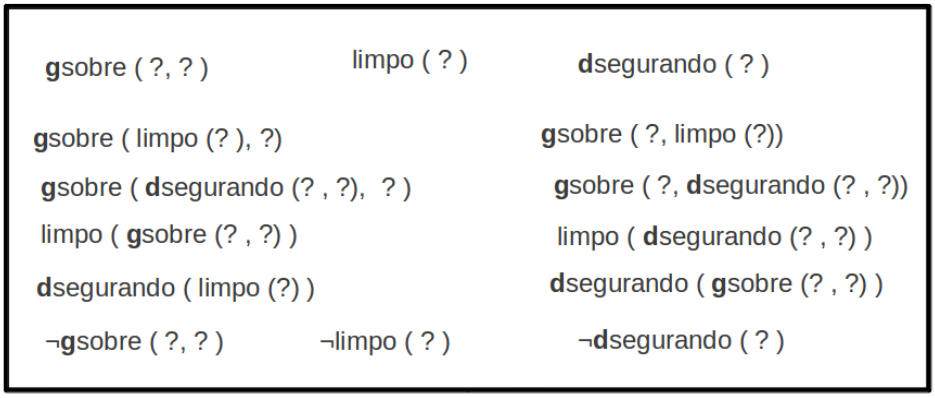

Figura 20 – *Exemplo de expansão de características: primeira expansão de um conjunto de características selecionadas na primeira execução da etapa de seleção de características, composto por **g**sobre(?, ?), clear(?) e **d**segurando (?) para o domínio Blocksworld.*

Agora assuma que na nova execução da etapa de seleção com as características obtidas na primeira expansão, as novas características selecionadas foram **g**sobre(limpo(?), ?) e ¬**d**segurando(?). Na Figura 21 são apresentados o conjunto de características resultado da segunda expansão. No exemplo utilizou-se os símbolos '&&' para representar a operação de intersecção resultado da aplicação da função de expansão **Especializacao**.

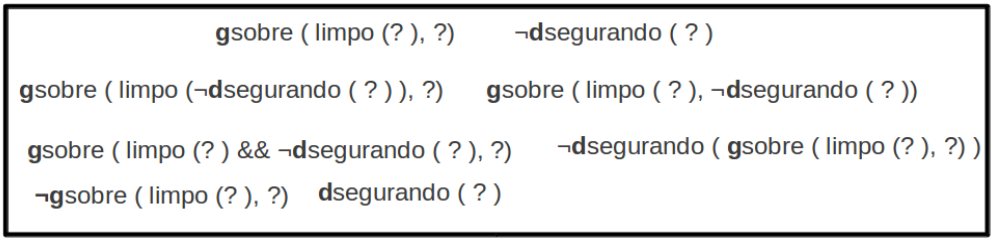

Figura 21 – *Exemplo de expansão de características: segunda expansão de um conjunto de características selecionadas na segunda execução da etapa de seleção de características, composto por **g**sobre(limpo(?), ?) e ¬**d**segurando(?), para o domínio Blocksworld.*

3.2 Limitações da abordagem de Yoon, Fern e Givan (2006, 2008)

A primeira consideração com relação ao trabalho de Yoon, Fern e Givan (2006, 2008) refere-se ao uso de combinações lineares na nova função heurística. Embora os resultados obtidos tenham sido bons, essa abordagem é útil apenas para a construção de modelos nos quais os parâmetros são lineares. Entretanto, em muitos problemas complexos, as expressões matemáticas que modelam o problema em questão são não lineares.

A segunda consideração é quanto ao método guloso utilizado na seleção de características. Esse método é mais eficiente quando existe pouco ruído nos dados e um número pequeno de características interagindo (VAFAIE; IMAM, 1994). Para os demais casos, faz-se necessária a escolha de uma abordagem de seleção de característica mais elaborada.

Dessa maneira, a Seção 3.3 discutirá uma abordagem alternativa desenvolvida neste trabalho, que explora as limitações apontadas acima.

3.3 Abordagem *LearnInPlanner*

A abordagem de aprendizado proposta neste trabalho consiste em modelar o problema de estimar os valores da função heurística definida por Yoon, Fern e Givan (2008) como um processo de regressão não linear via rede neural MLP. Neste caso, as entradas da rede representam a cardinalidade de um conjunto de características selecionadas por meio de um algoritmo genético.

É possível utilizar modelos de rede MLP com várias camadas ocultas, com diferentes quantidades de neurônios em cada camada. Para simplificar a escolha da arquitetura, optou-se, neste trabalho, por modelos com apenas uma camada oculta e fixou-se o número de neurônios nessa camada em função da dimensão das camadas de entrada e saída da rede. Para isso, o número de neurônios foi definido como a média aritmética entre o tamanho da entrada e o tamanho da saída da rede, tal como sugerido em Bishop (1996).

Dessa maneira, consegue-se evitar que sejam utilizadas unidades demais, o que pode levar a rede a memorizar os dados de treinamento (*overfitting*), prejudicando a generalização do modelo para dados ainda não vistos. E ainda, evitar que esse número seja muito pequeno, o que pode fazer com que a rede demore para encontrar uma representação adequada para o problema aprendido (BISHOP, 1996). Para exemplificar, a Figura 22 apresenta uma arquitetura de rede para um caso em que se considera cinco características

como entrada, um neurônio na camada de saída e que, conforme calculado por meio da média aritmética, possui três neurônios na camada oculta.

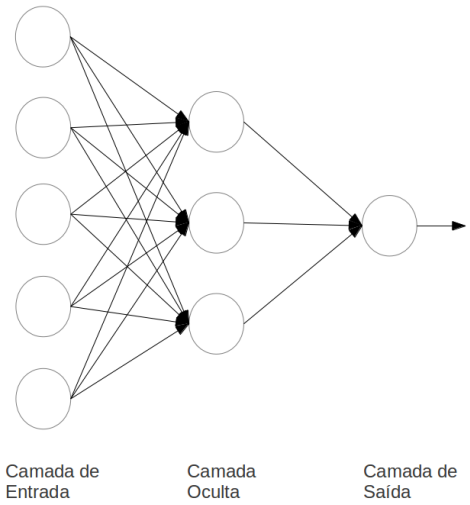

Figura 22 – *Exemplo de Arquitetura de Rede MLP.*

Segundo Bates e Watts (2007), a regressão não linear é uma técnica robusta para aproximação de funções e, portanto, seria mais apropriada para modelos não lineares. Portanto, com o uso da regressão não linear espera-se que a estimativa final seja mais precisa do que a obtida por meio de técnicas lineares.

Na abordagem proposta no presente trabalho, o método de seleção de características guloso utilizado em Yoon, Fern e Givan (2006, 2008) foi substituído por uma abordagem de algoritmo genético (GA). Segundo Vafaie e Imam (1994), o GA fornecem uma solução mais robusta para lidar com os dados, em comparação a abordagem gulosa, embora levem ao aumento do esforço computacional para solução do problema.

A principal característica do GA é a habilidade em explorar a informação de maneira cumulativa. Isso significa que o algoritmo parte de um espaço de busca inicial desconhecido, mas nas buscas subsequentes, os subespaços explorados tendem a ser mais promissores. Dessa maneira, a natureza do GA permite combinações de conjuntos de características que talvez não fossem testadas pelo método guloso e que podem levar a melhores resultados (VAFAIE; IMAM, 1994).

Para a utilização da abordagem de GA para a seleção de características mostrou-se necessária a definição de uma representação cromossômica binária, em que cada posição do cromossomo representa a presença ou ausência de cada uma das N característicias,

conforme proposto por Vafaie e Imam (1994). No decorrer da execução, cada uma das soluções produzidas pelo GA é avaliada por uma função de aptidão que corresponde à acurácia de uma rede neural MLP para um conjunto de validação em que somente as características apontadas pelo cromossomo são consideradas. Portanto, a entrada para cada modelo de rede MLP serão os valores de cardinalidade para as características definidas como presentes pelo cromossomo correspondente. À medida em que o GA evolui iterativamente, o processo de seleção de características desenvolve-se simultaneamente com o treinamento da respectiva rede MLP, de modo que, ao final da sua execução, a melhor solução fornece o modelo neural treinado com as características selecionadas.

Assim como na abordagem de Yoon, Fern e Givan (2008), o processo de aprendizado aqui proposto é alternado por uma etapa de expansão de características que consiste em redimensionar as soluções iniciais (cromossomos) do GA para acomodar um número maior de características. Na prática, isso significa executar novamente o GA sob essas novas configurações. Esse processo de aprendizado/expansão continua até que um número máximo de expansões seja atingido ou um desempenho mínimo seja alcançado pelo modelo de regressão.

Dessa forma, o procedimento proposto para aprendizado da função heurística é esquematizado nos seguintes passos:

1. Modelar o problema de seleção de características via GA considerando apenas características de profundidade 0 ou 1. Cada solução é avaliada pela acurácia obtida pelo modelo de rede MLP condizente com o respectivo cromossomo.

2. Executar o GA até que um número máximo de gerações seja atingido ou uma solução satisfatória seja encontrada.

3. Realizar a expansão do conjunto de características e a subsequente remodelagem do problema de seleção via GA. Nessa fase, os cromossomos são redimensionados para abrigar um número maior de características.

4. Executar o GA com as mesmas condições de parada da etapa 2.

5. Repetir os passos 3 e 4 até que o número máximo de expansões seja alcançado ou uma solução satisfatória seja encontrada.

A Figura 23 resume a abordagem de aprendizado para problemas de planejamento proposta neste trabalho. Para avaliar o desempenho da heurística calculada foi empregado

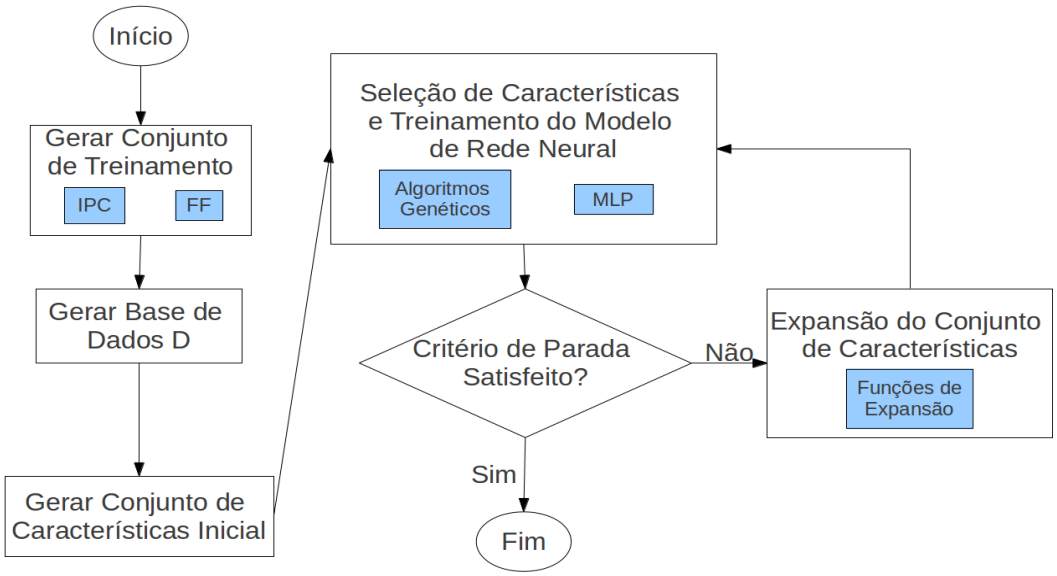

Figura 23 – *Esquema da Abordagem LearnInPlanner.*

o algoritmo de Busca Gulosa pela Melhor Escolha[1], conforme proposto em Yoon, Fern e Givan (2008), que avalia os nós na árvore ou grafo de busca por meio da função heurística definida. Uma vez que os algoritmos gulosos nem sempre produzem soluções ótimas (CORMEN et al., 2002), com a melhora na qualidade da heurística utilizada é possível obter uma melhoria na qualidade do resultado do algoritmo de busca.

[1]Do original, em inglês, *Greedy Best-First Search*.

Capítulo 4

Discussão dos resultados

Este capítulo apresenta a discussão dos resultados obtidos nos experimentos deste trabalho. Para isso, foi realizada uma análise comparativa entre a execução do algoritmo de busca gulosa apenas com a heurística do plano relaxado (abrevida como **HPR** para simplificar) e o uso da heurística do plano relaxado acrescida do valor estimado pelo modelo de rede neural MLP aprendido com as características selecionadas (referida como heurística **HPR+L**).

Para os experimentos foram considerados cinco domínios da Competições Internacionais de Planejamento IPC2[1] (2000) e IPC3(2002)[2]: *Blocksworld*, *Zenotravel*, *Rovers*, *Depots* e *Freecell*. Vale observar que os domínios *Blocksworld*, *Zenotravel*, *Rovers* são considerados mais simples para a tarefa de planejamento em comparação aos domínios *Depots* e *Freecell* (YOON; FERN; GIVAN, 2008).

A Seção 4.1 apresenta os detalhes relevantes da implementação da abordagem *LearnInPlanner*. A Seção 4.2 discute o procedimento realizado para a escolha dos parâmetros do modelo de rede neural MLP e da abordagem de algoritmos genéticos (GA) utilizada. As Seções 4.3, 4.4, 4.5, 4.6 e 4.7 apresentam a discussão dos resultados dos experimentos para os domínios *Blocksworld*, *Rovers*, *Zenotravel*, *Depots* e *Freecell*, respectivamente. Por fim, a Seção 4.8 contém as considerações finais referente aos experimentos realizados.

4.1 Detalhes de implementação

O planejador *LearnInPlanner* foi desenvolvido utilizando a linguagem de programação Java, com a *Java Development Kit* versão 7. A implementação da rede MLP teve como base o código da máquina de aprendizado RBM disponível em (JARBM, 2011), pois essa implementação possui um conjunto de classes prontas representando camadas de neurônios úteis para construir modelos de redes neurais com diferentes funções de ativação.

Para geração do conjunto de treinamento para o algoritmo de aprendizado foi uti-

[1]Do original, em inglês, *International Planning Competition*.
[2]Informações e domínios disponíveis em http://ipc.icaps-conference.org/.

lizada uma implementação do planejador FF, desenvolvida em C, disponível em (FAST-FORWARD, 2011) sob a licença GNU de *software* livre. A implementação da heurística do plano relaxado e do algoritmo de busca gulosa foi desenvolvida em (JAVAFF, 2008), também sob a licença GNU de *software* livre.

É importante ressaltar que a implementação disponibilizada em (JAVAFF, 2008) apresentava problemas de desempenho no que se refere ao uso de memória, o que demandou tempo de estudo e implementação a fim de otimizar o algoritmo. Foram feitas alterações nos procedimentos de instanciação dos objetos, de maneira a alocar espaços de memória somente quando necessário.

Uma vez que a 'limpeza' da memória em java é de responsabilidade do *Garbage colector* da própria linguage, foi preciso inserir uma lógica para realizar a execução desse procedimento sob demanda. Dessa maneira, os objetos não mais em uso são removidos da memória o mais cedo possível. Essas alterações possibilitaram controlar o consumo de memória do planejador *LearnInPlanner* que, para os domínios testados no presente trabalho, girou em torno de 4GB de RAM.

Para os experimentos com os domínios *Blocksworld* e *Zenotravel* foi utilizado um computador *desktop* com processador Intel Core i7-3770 CPU 3.40GHz, da família Ivy Bridge. Por sua vez, um computador *desktop* com processador Intel Core i7-2700K CPU 3.50GHz, família Sandy Bridge, foi utilizado nos experimenos com os domínios *Rovers*, *Depots* e *Freecell*. Ambos os computadores com 16GB de memória RAM e com sistema operacional linux ubuntu 12.04, versão 64-bits.

4.2 Escolha de parâmetros do modelo de rede MLP e da abordagem de algoritmos genéticos (GA)

Tanto a rede MLP quanto a abordagem de GA possuem vários parâmetros que precisam ser definidos empiricamente para o aprendizado. Uma vez que testar todas as combinações possíveis de parâmetros é impraticável, alguns valores foram mantidos fixos para todos os domínios de problemas de planejamento. São eles:

- Número de indivíduos escolhidos para participar do processo de reprodução do GA a cada geração igual a 6;
- Número de pontos nas operações de cruzamento do GA definido como 1;
- Taxa de mutação do GA igual a 0,1;

- Número máximo de gerações igual a 10;
- Erro quadrático médio desejável igual a 0,001. Nesse caso, tal medida corresponde ao valor da função de aptidão de cada cromossomo;
- Taxa de aprendizado dos modelos de rede neural MLP foi definido como 0.1.

Considera-se que o GA convergiu se o número máximo de gerações ou o erro desejável foi atingido. Para o treinamento dos modelos de rede MLP foi utilizado o número máximo de épocas como critério de parada. Para os domínios de planejamento *Blocksworld* e *Rovers* definiu-se o número máximo de épocas igual a 100, pois, por serem domínios mais simples, necessitam de menos ajustes nos pesos do modelo de rede MLP para convergir para uma solução satisfatória. Por sua vez, o domínio *Freecell* precisou de 500 épocas para convergir. Por fim, os domínios *Zenotravel* e *Depots* utilizaram, nos experimentos, o limite máximo de 2000 épocas.

4.3 Experimentos com o domínio *Blocksworld*

O domínio *Blocksworld* disponibiliza 35 problemas para planejamento, dentre os quais foram separados 28 para a etapa de aprendizado e os 7 mais complexos para etapa de planejamento. Dos 28 problemas para aprendizado, 25 foram utilizados no treinamento dos modelos de rede MLP e 3 foram separados em um conjunto de validação utilizado para calcular o valor da função de aptidão dos cromossomos do GA. Os resultados da etapa de aprendizado encontram-se resumidos na Tabela 1, contendo: o erro quadrático médio obtido pelo modelo de rede MLP treinado utilizando as características selecionadas; o número de expansões no conjunto de características necessário para selecionar as características mais relevantes para a construção da função heurística; o total de características da última expansão realizada; e o número de características selecionadas no processo de aprendizado.

Tabela 1 – *Resultados da etapa de aprendizado para o domínio Blocksworld.*

Erro Quadrático Médio	Nº de Expansões de Características	Total de Características	Nº de Características Selecionadas
0.0006	2	1902	950

Para avaliar o desempenho do modelo aprendido juntamente com as características selecionadas, foram realizados experimentos de planejamento com os problemas separados

para teste. Neste caso, foi estabelecido um limite de tempo máximo em 24 horas, após o que, considera-se que houve falha no planejamento. A Tabela 2 resume os resultados obtidos utilizando a heurística **HPR** em comparação com os resultados utilizando a heurística **HPR+L**. Nessa tabela, o total de nós visitados refere-se ao número de estados visitados no grafo de busca; os nós avaliados são os estados efetivamente avaliados, excluindo-se os estados visitados mais de uma vez.

Tabela 2 – *Resultados da etapa de planejamento para o domínio Blocksworld. Em destaque o resultado em que a heurística HPR+L encontra um plano menor do que a heurística HPR.*

Nº do Problema	Heurística	Nós avaliados	Total de nós visitados	Tamanho do plano	Tempo
29	**HPR**	863	1155	48	0,31min
29	**HPR+L**	1199	2376	96	21,25min
30	**HPR**	30	30	28	0,01min
30	**HPR+L**	39	40	32	1,51min
31	**HPR**	66	75	40	0,06min
31	**HPR+L**	1269	2104	46	51,27min
32	**HPR**	22	22	20	0,02min
32	**HPR+L**	31	31	26	3,04min
33	**HPR**	52	53	42	0,11min
33	**HPR+L**	7985	15551	198	187,61min
34	**HPR**	15	15	14	0,04min
34	**HPR+L**	23	23	18	3,57min
35	**HPR**	31	31	28	0,12min
35	**HPR+L**	27	27	26	7,5min

Observa-se que ambas as heurísticas conseguiram resolver todos os problemas escolhidos para o domínio *Blocksworld*. Neste caso, o uso da heurística **HPR** apresentou melhores resultados no que se refere ao tamanho do plano, que, com exceção do problema 35, foi menor em comparação à utilização da **HPR+L**. Esse resultado indica que o uso da abordagem não-linear pode ter sido uma escolha de abordagem muita complexa para solucionar os problemas mais simples do *Blocksworld*.

Os resultados obtidos por Yoon, Fern e Givan (2008) reforçam essa hipótese, conforme mostrado na Tabela 3, em que **H-Yoon** é a nomenclatura definida para representar a nova função heurística construída pela abordagem linear. Nesse trabalho relacionado, para o domínio *Blocksworld*, foram utilizados 15 problemas para o aprendizado e 20 para o planejamento. O limite de tempo para o planejamento utilizado foi de 30 minutos. Como pode-se observar, a abordagem de aprendizado linear foi suficiente para melhorar o

desempenho do planejador com relação ao tamanho do plano. O tempo de planejamento também não aumentou consideravelmente.

Tabela 3 – *Resultados obtidos por Yoon, Fern e Givan (2008) para o domínio Blocksworld.*

Heurística	Nº Problemas Solucionados	Tempo médio de planejamento	Tamanho médio do plano	Nós avaliados
HPR	20	11.74sec	116	12932
H-Yoon	20	12.94sec	82.7	31940

Um outro comportamento que merece destaque nos experimentos realizados neste trabalho refere ao tempo de planejamento que, para todos os casos testados, foi consideravelmente maior quando acrescida a correção estimada pelo modelo aprendido na heurística do plano relaxado. Esse tempo maior demonstra que, com a heurística **HPR+L**, o planejador precisou de mais tempo para avaliar cada nó no grafo de busca. Nesse caso, avaliar um nó compreende calcular a estimativa de custo para atingir o estado meta para todos os caminhos possíveis a partir de um determinado estado e, de acordo com o valor calculado, escolher por qual dos caminhos a busca irá seguir.

Dessa maneira, o fato de a heurística **HPR+L** demorar mais tempo para avaliar cada nó pode ser resultado não apenas da inserção das informações adicionais, que resulta em mais cálculos a serem realizados, mas também pode indicar que, com a heurística **HPR+L**, o algoritmo de busca tende a seguir por caminhos em que cada estado a ser avaliado possuía um maior número de caminhos possíveis para atingir o estado meta em comparação ao caminho seguido pelo algoritmo quando guiado pela heurística **HPR**.

4.4 Experimentos com o domínio *Rovers*

O domínio *Rovers* possui 20 problemas de planejamento, numerados de 1 a 20 em ordem de complexidade em termos de tamanho do plano e tempo necessário para o planejamento. Para os experimentos, os problemas foram divididos em dois conjuntos, sendo 15 para aprendizado e os 5 mais complexos para teste com o planejador. Dos 15 problemas para aprendizado, 13 foram utilizados no treinamento dos modelos de rede MLP e 2 foram separados em um conjunto de validação utilizado para calcular do valor da função de aptidão dos cromossomos do GA. Os resultados obtidos na fase de aprendizado são mostrados na Tabela 4.

Tabela 4 – *Resultados da etapa de aprendizado para o domínio Rovers.*

Erro Quadrático Médio	Nº de Expansões de Características	Total de Características	Nº de Características Selecionadas
0.0012	0	134	72

Em comparação ao domínio *Blocksworld*, discutido na Seção 4.3, o domínio *Rovers* mostrou ser um domínio mais complexo na etapa de aprendizado, apresentando um erro maior na estimativa do novo valor de heurística. Observa-se ainda que o conjunto de características selecionados antes das expansões foi o que obteve melhor desempenho no treinamento do modelo de rede MLP, em comparação com as expansões realizadas.

A Tabela 5 apresenta os resultados do planejamento para o domínio *Rovers*, com destaque para os problemas solucionados em que a heurística **HPR+L** obteve um plano de tamanho menor em comparação a heurística **HPR**. Aqui, cabe observar que os experimentos foram iniciados com limite de tempo igual a 30 minutos e, em caso de falha do planejador, esta medida foi progressivamente aumentada na tentativa de resolver os outros problemas mais complexos. Assim, para o problema não solucionado é mostrada apenas a execução com limite de tempo maior utilizado.

Tabela 5 – *Resultados da etapa de planejamento para o domínio Rovers. Em destaque os problemas solucionados que possuem tamanho de plano menor para a heurística **HPR+L**.*

Nº do Problema	Heurística	Nós avaliados	Total de Nós Visitados	Limite de Tempo	Tamanho do plano	Tempo
16	**HPR**	888	1266	30min	44	0,75min
16	**HPR+L**	83	88	30min	65	18,35min
17	**HPR**	289	373	30min	60	3,35min
17	**HPR+L**	61	61	3h	56	88,76min
18	**HPR**	191	242	30min	48	9,84min
18	**HPR+L**	169	200	12h	74	663,4min
19	**HPR**	7684	10455	12h	94	442,41min
19	**HPR+L**	90	97	24h	67	750,51min
20	**HPR**	32294	44411	158h	-	-
20	**HPR+L**	124	140	48h	-	-

Conforme observa-se, de maneira análoga ao domínio *Blocksworld*, com a utilização da **HPR+L**, cada nó necessitou de mais tempo para ser avaliado. Contudo, para os problemas 17 e 19, esse tempo adicional de planejamento foi parcialmente compensado

com um plano solução de tamanho menor quando comparado aos planos encontrados com a heurística **HPR**.

Outro fato a se observar é, que para todos os problemas solucionados com a heurística **HPR+L**, foi necessário visitar menos estados para encontrar o plano solução e a quantidade de estados visitados repetidos foi significativamente menor, em comparação à utilização da heurística **HPR**.

Com relação ao problema 20, o mais complexo entre os testados, ambas as heurísticas, **HPR** e **HPR+L**, mostraram-se incapazes de solucioná-lo dentro de um limite de tempo aceitável. Para a heurística **HPR**, o limite de tempo foi aumentado até atingir 158 horas, quando considerou-se que houve falha no planejamento. Uma vez que com a heurística **HPR+L**, para todos os outros casos, o tempo de planejamento necessário foi maior, e com a heurística **HPR**, nem em 158 horas foi possível encontrar solução para o problema 20, considerou-se que houve falha no planejamento também para a heurística **HPR+L**.

Por fim, vale ressaltar que uma vez que o domínio *Rovers* foi considerado simples segundo Yoon, Fern e Givan (2008), este não foi utilizado em seus experimentos.

4.5 Experimentos com o domínio *Zenotravel*

Similarmente ao domínio *Rovers*, o domínio *Zenotravel* possui 20 problemas de planejamento e, esses foram foram divididos em dois conjuntos, sendo 15 para aprendizado e os 5 mais complexos para teste com o planejador. Dos 15 problemas para aprendizado, 13 foram utilizados no treinamento dos modelos de rede MLP e 2 foram separados em um conjunto de validação utilizado para calcular do valor da função de aptidão dos cromossomos do GA. A Tabela 6 resume os resultados obtidos no aprendizado do domínio *Zenotravel*.

Tabela 6 – *Resultados da etapa de aprendizado para o domínio Zenotravel.*

Erro Quadrático Médio	Nº de Expansões de Características	Total de Características	Nº de Características Selecionadas
0.0012	1	128	53

Com relação ao conjunto de características selecionadas, devido ao conjunto de características inicial do domínio *Zenotravel* possuir menos características, no caso 25, em comparação ao domínio *Rovers*, que possuia 134, o número de características selecionadas

para o domínio *Zenotravel* foi menor, mesmo realizando-se uma expansão no conjunto de características. Entretanto, observa-se que para esse domínio o erro no aprendizado foi o mesmo do domínio *Rovers* (ver Tabela 4) com esse conjunto menor de características.

A Tabela 7 mostra os resultados do planejamento do domínio *Zenotravel* com as heurísticas **HPR** e **HPR+L**, com destaque para os problemas em que o planejador conseguiu encontrar um plano solução menor com a heurística **HPR+L** em comparação ao tamanho do plano solução quando utilizada a heurística **HPR**. Para esse domínio, os limites de tempo utilizados para o planejamento foram superiores aos considerados para o domínio *Rovers*.

Dessa maneira, iniciou-se os experimentos com um limite de tempo igual a 24 horas e, para os casos de falha no planejamento, esse valor foi aumentado progressivamente até o problema ser solucionado ou atingir um limite de tempo máximo aceitável para o planejamento. Ressalta-se ainda que para os problemas não solucionados, na Tabela 7, é apresentado o resultado para o maior limite de tempo utilizado para planejamento.

Tabela 7 – *Resultados da etapa de planejamento para o domínio Zenotravel. Em destaque os problemas solucionados que possuem tamanho de plano menor para a heurística **HPR+L**.*

Nº do Problema	Heurística	Nós avaliados	Total de Nós Visitados	Limite de Tempo	Tamanho do plano	Tempo
16	**HPR**	75	76	24h	65	194min
16	**HPR+L**	58	59	120h	53	4656min
17	**HPR**	51	51	24h	50	1196min
17	**HPR+L**	94	98	200h	72	10593min
18	**HPR**	189	221	48h	100	476min
18	**HPR+L**	82	82	120h	78	6614min
19	**HPR**	123	134	158h	93	809min
19	**HPR+L**	50	59	200h	-	-
20	**HPR**	301	349	158h	97	6313min
20	**HPR+L**	29	29	200h	-	-

Conforme já foi observado nos domínios testados anteriormente, para o domínio *Zenotravel*, o planejamento com a heurística **HPR+L** demanda mais tempo do que o planejamento com a heurística **HPR**, pois o tempo necessário para avaliar cada nó no grafo de busca tende a ser maior com a heurística **HPR+L**. Contudo, para os dois dos três problemas solucionados por ambas as heurísticas, mesmo gastando maior tempo para solução, a heurística **HPR+L** obteve um plano com tamanho menor em comparação ao plano obtido com a heurística **HPR**.

Outro comportamento observado é que o tempo necessário para encontrar o plano solução dos problemas de planejamento do domínio *Zenotravel* foi consideravelmente maior do que o tempo utilizado pelos domínios mais simples, *Blocksworld* e *Rovers*. Por fim, vale ressaltar que, de maneira similar ao domínio *Rovers*, o domínio *Zenotravel* não foi utilizado nos experimentos realizados por Yoon, Fern e Givan (2008), por ser considerado um domínio simples.

4.6 Depots

O domínio Depots possui 21 problemas para planejamento, numerados de 1 a 21 em ordem de complexidade em termos de tamanho do plano e tempo necessário para planejamento. De maneira análoga a Yoon, Fern e Givan (2008), foram separados os 5 problemas de planejamento mais complexos para os testes e os 16 demais problemas foram utilizados no aprendizado. Dos 16 problemas separados para o aprendizado, 14 foram utilizados no treinamento dos modelos de rede neural MLP e 2 foram utilizados no conjunto de validação para o cálculo do valor da função de aptidão dos cromossomos do GA.

A Tabela 8 apresenta os resultados da etapa de aprendizado. Conforme pode ser observado, o erro apresentado pelo domínio *Depots* é menor em comparação ao erro apresentado pelo domínios *Rovers* e *Zenotravel* e, apenas um pouco maior que o erro obtido pelo domínio *Blocksworld*.

Tabela 8 – *Resultados da etapa de aprendizado para o domínio Depots.*

Erro Médio Quadrático	Nº de Expansões de Características	Total de Características	Nº de Características Selecionadas
0.0007	1	195	93

Para a fase de planejamento, realizou-se testes com diferentes limites de tempo, aumentados progressivamente na tentativa de resolver os problemas. Uma vez que o domínio *Depots* é mais complexo que os domínios anteriores (YOON; FERN; GIVAN, 2008), apenas os problemas 17 e 18 foram considerados nos experimentos, por serem os mais simples. A Tabela 9 apresenta os resultados obtidos na fase de planejamento, com destaque para os problemas solucionados e, no caso dos não solucionados, foi apresentado o resultados para o maior limite de tempo utilizado.

Tabela 9 – *Resultados da etapa de planejamento para o domínio Depots.*

Nº do Problema	Heurística	Nós executados	Total de Nós Visitados	Limite de Tempo	Tamanho do plano	Tempo
17	**HPR**	23	24	30min	23	5min
17	**HPR+L**	753	1019	200h	-	-
18	**HPR**	2651	3787	72h	76	2462min
18	**HPR+L**	288	395	200h	-	-

Com exceção do problema 17 que foi solucionado com limite de tempo baixo com a heurística **HPR**, os testes com os problemas 17 e 18 demonstram necessitar de um tempo maior para o planejamento tanto com a heurística **HPR** quanto com a heurística **HPR+L**. O problema 18, por exemplo, utilizando um limite de tempo de 72 horas, foi solucionado em aproximadamente 41 horas utilizandos a heurística **HPR**. Para a heurística **HPR+L**, o limite de tempo de 200 horas não foi suficiente para o planejador encontrar uma solução.

Como comparativo, a Tabela 10 apresenta os resultados obtidos por Yoon, Fern e Givan (2008) para o domínio *Depots* com limite de tempo para planejamento de 30 minutos. De forma análoga ao presente trabalho, quando utilizado o limite de tempo para planejamento de 30 minutos, o planejador foi capaz de solucionar apenas um problema com heurística **HPR**. Em contrapartida, a heurística **H-Yoon** foi capaz de solucionar todos os problemas separados para planejamento, embora com um tamanho médio de plano solução maior quando comparado ao plano solução com a heurística **HPR**.

Tabela 10 – *Resultados obtidos por Yoon, Fern e Givan (2008) para o domínio Depots.*

Heurística	Nº Problemas Solucionados	Tempo médio de planejamento	Tamanho médio do plano	Nós avaliados
HPR	1	0.28sec	29	1106
H-Yoon	5	174sec	68.4	71795

4.7 Experimentos com o domínio *Freecell*

Para os experimentos, os 20 problemas de planejamento disponíveis para o domínio *Freecell* foram separados em 15 para aprendizado e 5 para teste com o planejador, de forma análoga a Yoon, Fern e Givan (2008). Os resultados da fase de aprendizado encontram-

se resumidos na Tabela 11. Nota-se que, para estimar a diferença entre o custo real para atingir o estado de meta e a heurística **HPR**, o aprendizado com o domínio *Freecell* utilizou um conjunto grande de características, de maneira similar ao domínio *Blocksworld*.

Tabela 11 – *Resultados da etapa de aprendizado para o domínio Freecell.*

Erro Médio Quadrático	Nº de Expansões de Características	Total de Características	Nº de Características Selecionadas
0.0012	1	1050	546

Para realizar o planejamento definiu-se limites de tempo maiores, uma vez que, conforme demonstrado com o domínio *Depots*, quanto mais complexo o problema de um determinado domínio, mais tempo o planejador tende a levar para encontrar um plano solução. Como nos experimentos anteriores, a utilização da heurística **HPR+L** demonstrou necessitar de maior tempo para o planejamento, os limites de tempo para planejamento considerados nos testes com essa heurística foram superiores aos limite de tempo considerados para a heurística **HPR**.

A Tabela 12 apresenta os resultados da etapa de planejamento para o domínio *Freecell*, com destaque para o problema que o planejador conseguiu encontrar um plano com tamanho menor quando o algoritmo de busca foi guiado pela heurística **HPR+L**. Para os demais problemas são apresentados os resultados obtidos com o limite de tempo maior considerado. Observa-se que, para esse domínio, o planejador apresentou maior dificuldade para solucionar os problemas com a heurística **HPR+L**, gastando um tempo excessivo para avaliar cada estado no grafo de busca. Conforme já observado em experimentos anteriores, os resultados indicam que com a utilização da heurística **HPR+L**, o planejador necessita de um tempo maior para avaliar cada nó no grafo de busca.

Além disso, com a heurística **HPR**, em um período de 200 horas, o planejador foi capaz de avaliar apenas 21 nós para o problema 18, enquanto para o problema 17 os nós avaliados foram apenas cinco. Uma vez que o aumento da complexidade do problema de planejamento considerado resulta em maior tempo necessário para planejamento, os demais problemas separados para os testes com o domínio *Freecell* não foram utilizados nos experimentos, pois, considerou-se que o tempo necessário para a tentativa de planejamento com esses problemas não seria computacionalmente aceitável.

Em termos de comparação, a Tabela 13 apresenta os resultados obtidos por Yoon, Fern e Givan (2008) para os problemas do domínio *Freecell* com um limite de tempo para

Tabela 12 – *Resutados da etapa de planejamento para o domínio Freecell. Em destaque o problema solucionado com tamanho de plano menor para a heurística* ***HPR+L****.*

Nº do Problema	Heurística	Nós executados	Total de Nós Visitados	Limite de Tempo	Tamanho do plano	Tempo
16	**HPR**	425	570	96h	86	1160min
16	**HPR+L**	82	83	158h	70	9128min
17	**HPR**	241	294	24h	102	939min
17	**HPR+L**	5	5	200h	-	-
18	**HPR**	2397	3424	200h	-	-
18	**HPR+L**	21	21	200h	-	-

planejamento igual a 30 minutos. Para esse domínio, a heurística **H-Yoon** apresentou, em média, planos de tamanho maior mas com um tempo médio necessário para planejamento ligeiramente menor. Além disso, a quantidade de nós avaliados com a heurística **H-Yoon** foi consideralmente menor do que a quantidade de nós avaliados pela heurística **HPR**.

Tabela 13 – *Resultados obtidos por Yoon, Fern e Givan (2008) para o domínio Freecell.*

Heurística	Nº Problemas Solucionados	Tempo médio de planejamento	Tamanho médio do plano	Nós avaliados
HPR	5	442sec	109	25678
H-Yoon	5	89sec	145	5096

4.8 Considerações Finais

Por meio dos experimentos apresentados anteriormente, pode-se concluir que a medida em que a complexidade do domínio aumenta, o planejador apresenta mais dificuldade para solucionar os problemas do domínio, tanto com a heurística **HPR** quanto com a heurística **HPR+L**. Neste trabalho, a complexidade de um problema foi medida pela grandeza da instância, ou seja, pela quantidade de objetos considerados no problema a ser solucionado.

Além disso, foi possível observar que a heurística proposta neste trabalho conseguiu resolver problemas dos três domínios mais simples considerados nos experimentos. Para o domínio *Blocksworld*, a heurística **HPR+L** encontrou um plano menor que o encontrado pela heurística **HPR** para o problema mais complexo considerado no conjunto de teste.

Por sua vez, para o domínio *Rovers*, a heurística **HPR+L** obteve planos menores do

que a **HPR** para dois dos cinco problemas considerados no conjunto de teste. O problema mais complexo nesse domínio não foi solucionado por ambas as abordagens, considerando-se o limite de tempo estabelecido. Já no caso do domínio *Zenotravel*, a heurística **HPR+L** obteve um plano menor, em comparação ao plano obtido pela heurística **HPR**, para os dois problemas solucionados dentro do limite de tempo estabelecido para os experimentos.

No que se refere ao tempo necessário para planejamento, o algoritmo de busca gulosa, utilizado neste trabalho, demonstrou demandar maior tempo para avaliar cada nó visitado quando guiado pela heurística **HPR+L**, em comparação ao tempo utilizado quando a busca foi guiada pela heurística **HPR**. Conforme explicado anteriormente, avaliar um nó requer realizar o cálculo do valor da heurística para todos os caminhos pelos quais é possível atingir o estado meta a partir do nó avaliado.

O tempo necessário para avaliar um nó pode ser influenciado por dois fatores. O primeiro fator refere-se ao tamanho do caminho para atingir o estado meta a partir do nó avaliado. Na medida em que o tamanho desse caminho aumenta, o tamanho do grafo construído para extração do plano relaxado tende a ser maior, portanto, demandando maior tempo necessário para a sua construção. O segundo fator, que influencia no tempo gasto para a avaliação dos nós do grafo de busca, consiste na quantidade de caminhos possíveis para atingir o estado meta a partir do nó considerado, pois, na medida em que a quantidade de caminhos aumenta, será necessário calcular o valor da heurística um maior número de vezes.

Esses fatores podem justificar porque o planejador falhou em solucionar os problemas de planejamento dos domínios mais complexos utilizados nos experimentos. Com problemas mais complexos para solucionar, o tamanho do grafo construído para extração do plano relaxado e, principalmente, a quantidade de caminhos para os quais é necessário o cálculo da heurística tendem a ser maiores. Isso torna inviável a aplicação da abordagem proposta, com a utilização do algoritmo de busca gulosa, para domínios complexos.

Capítulo 5

Conclusões e Trabalhos Futuros

Nesta dissertação, foi apresentada a abordagem *LearnInPlanner* que, com o auxílio de técnicas de aprendizado de máquina, inclui novas informações no cálculo da heurística do planejador com o objetivo de melhorar sua precisão para solução de problemas de planejamento clássico. Esse planejador utiliza um algoritmo de busca heurística progressiva para o planejamento automatizado. A função heurística estimada tem como base a heurística do planejador FF e é estimada por meio de um processo de regressão não linear.

Um dos diferenciais do presente trabalho é a inclusão de informações adicionais, representadas por meio de características do problema, à heurística com o objetivo de melhorar sua precisão. O principal objetivo deste trabalho foi explorar as limitações de algumas abordagens atuais de planejamento baseado em busca heurística progressiva e contribuir com a avaliação de técnicas de aprendizado de máquina que possam ajudar a superar algumas dessas limitações. Para realizar o processo de regressão não linear foi utilizada uma rede neural MLP. A presente abordagem inclui ainda a utilização de uma abordagem de algoritmos genéticos (GA) para seleção das características agregadas à função heurística.

Conforme observado nos experimentos, a abordagem *LearnInPlanner* apresentou dificuldade para solucionar os problemas de planejamento mais complexos dentro de um tempo computacionalmente aceitável. Como tentativa de diminuir essa limitação, tem-se como possível trabalho futuro a tradução da implementação da abordagem aqui desenvolvida para uma linguagem de programação mais eficiente no que se refere ao tempo de processamento.

Outro trabalho futuro consiste no estudo do processo de seleção de características, com variações tanto na quantidade de gerações do GA quanto na quantidade de expansões do conjunto de características para avaliar a influência desses fatores no desempenho no planejamento. O intuito seria realizar uma análise comparativa entre a utilização de um número variável de características para o cálculo do valor da heurística e uma análise do desempenho de características de diferentes profundidades na solução dos problemas de planejamento. Ainda no que se refere ao processo de seleção de características, um trabalho interessante seria um estudo sobre possíveis metodologias para avaliação da qua-

lidade de um conjunto de características selecionadas, dado um determinado domínio de planejamento.

Neste trabalho foi utilizada a busca gulosa como algoritmo de busca para o planejamento, mas existem vários outros algoritmos que podem ser utilizados para esse fim. Nesse sentido, um estudo interessante refere-se à substituição do algoritmo guloso por algoritmos mais eficiente de busca que possam melhorar o desempenho do planejador com relação ao tempo de execução. A abordagem de cálculo de heurística aqui proposta poderia, por exemplo, ser agregada ao planejador FF, que utiliza uma variação do algoritmo de busca de subida de encosta, denominada *Enforced Form Hill-Climbing*, que combina busca local e sistemática para a realização do planejamento. Para melhorar a eficiência do algoritmo de busca, o planejador FF utiliza ainda duas técnicas de poda. A primeira técnica consiste em selecionar um conjunto de sucessores promissores para cada nó no grafo de busca. Por sua vez, a segunda técnica procura eliminar ramos do grafo de busca onde o objetivo aparenta ter sido atingido muito cedo.

Referências

BACK, T.; FOGEL, D. B.; MICHALEWICZ, T. *Evolutionary Computation 1: Basic Algorithms and Operators*. 1st. ed. : Taylor & Francis, 2000. Paperback. ISBN 0750306645.

BATES, D. M.; WATTS, D. G. *Nonlinear Regression Analysis and Its Applications*. 1. ed. : Wiley, 2007.

BISHOP, C. M. *Neural Networks for Pattern Recognition*. 1. ed. : Oxford University Press, USA, 1996. Paperback. ISBN 9780198538646.

BLUM, A. L.; FURST, M. L. Fast planning through planning graph analysis. *Artificial Intelligence*, v. 90, n. 1, p. 1636–1642, 1997.

BONET, B.; GEFFNER, H. *HSP: Heuristic Search Planner*. 1998.

BONET, B.; GEFFNER, H. Planning as heuristic search: New results. In: *IN PROCEEDINGS OF ECP-99*. : Springer, 1999. p. 360–372.

BONET, B.; GEFFNER, H. Hsp: Heuristic search planner. *AI Magazine*, v. 21, 2000.

BONET, B.; GEFFNER, H. Heuristic search planner 2.0. *AI Magazine*, v. 22, p. 77–80, 2001.

BONET, B.; LOERINCS, G.; GEFFNER, H. A robust and fast action selection mechanism for planning. In: *In Proceedings of AAAI-97*. : MIT Press, 1997. p. 714–719.

CORMEN, T. H. et al. *Algoritmos: teoria e prática*. 6. ed. : Editora Campus, 2002.

FAST-FORWARD. 2011. Disponível em: <http://www.loria.fr/ hoffmanj/ff.html>.

FAUSETT, L. (Ed.). *Fundamentals of neural networks: architectures, algorithms, and applications*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1994. ISBN 0-13-334186-0.

FIKES, R. E.; NILSSON, N. J. Strips: a new approach to the application of theorem proving to problem solving. In: *Proceedings of the 2nd international joint conference on Artificial intelligence*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1971. p. 608–620.

GHALLAB, M.; NAU, D.; TRAVERSO, P. *Automated Planning: Theory and Practice*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2004. ISBN 1558608567.

HAYKIN, S. *Redes Neurais: Princípios e Prática*. 2. ed. : Bookman, 2007.

HOFFMANN, J. Ff: The fast-forward planning system. *AI Magazine*, v. 22, n. 3, p. 57–62, 2001.

HOFFMANN, J.; NEBEL, B. The ff planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, v. 14, p. 263–302, 2001.

HOLLAND, J. H. *Adaptation in Natural and Artificial Systems*. Ann Arbor, MI, USA: University of Michigan Press, 1975.

JARBM. 2011. Disponível em: <http://sourceforge.net/projects/jarbm/>.

JAVAFF. 2008. Disponível em: <http://www.inf.kcl.ac.uk/staff/andrew/JavaFF/>.

JONG, K. D. Adaptive system design: A genetic approach. *IEEE Transactions on Systems, Man and Cybernetics*, v. 10, n. 9, p. 566–574, set. 1980.

JONG, K. D. Learning with genetic algorithms: An overview. *Machine Learning*, v. 3, n. 2-3, out. 1988.

MCALLESTER, D.; GIVAN, R. Taxonomic syntax for first order inference. *Journal of the Association for Computing Machinery*, ACM, New York, NY, USA, v. 40, n. 2, p. 246–283, April 1993. ISSN 0004-5411.

MCDERMOTT, D. et al. *PDDL - The Planning Domain Definition Language*. 1998.

MCDERMOTT, D. V.; HENDLER, J. A. Planning: What it is, what it could be, an introduction to the special issue on planning and scheduling. *Artificial Intelligence*, v. 76, n. 1-2, p. 1–16, 1995.

MITCHELL, T. M. *Machine Learning*. 1. ed. New York, NY, USA: McGraw-Hill, Inc., 1997. ISBN 0070428077, 9780070428072.

ORPONEN, P. Computational complexity of neural networks: A survey. *Nordic Journal of Computing*, September 1994. ISSN 1879-1298.

RUMELHART, D.; HINTONT, G.; WILLIAMS, R. Learning representations by back-propagating errors. *Nature*, v. 323, n. 6088, p. 533–536, 1986.

RUSSELL, S.; NORVIG, P. *Artificial Intelligence: A Modern Approach (2nd Edition)*. 2. ed. : Prentice Hall, 2002. Hardcover. (Prentice Hall series in artificial intelligence). ISBN 0137903952.

VAFAIE, H.; IMAM, I. F. I.: feature selection methods: Genetic algorithms vs greedy-like search. In: *In: Proceedings of the International Conference on Fuzzy and Intelligent Control Systems*. 1994.

VAFAIE, H.; JONG, K. D. Genetic algorithms as a tool for feature selection in machine learning. In: . : Society Press, 1992. p. 200–204.

VIDAL, V. A lookahead strategy for heuristic search planning. In: *Proceedings of the 14th International Conference on Automated Planning and Scheduling (ICAPS 04)*. Whistler, BC, Canada: AAAI Press, 2004. p. 150–159.

WELD, D. S. Recent advances in ai planning. *AI Magazine*, v. 20, n. 2, p. 93–123, 1999.

WILKINS, D. E. Domain-independent planning: Representation and plan generation. *Artificial Intelligence*, v. 22, n. 3, p. 269 – 301, 1984. ISSN 0004-3702.

YOON, S.; FERN, A.; GIVAN, R. Learning heuristic functions from relaxed plans. In: *International Conference on Automated Planning and Scheduling Automated Planning and Scheduling*. 2006. p. 162–171.

YOON, S.; FERN, A.; GIVAN, R. Learning control knowledge for forward search planning. *Journal of Machine Learning Research*, v. 9, p. 683–718, 2008.